



TECHNISCHE UNIVERSITÄT  
BERGAKADEMIE FREIBERG

The University of Resources. Since 1765.

# Spatio-temporal Information System for the Geosciences: Concepts, Data models, Software, and Applications

By the Faculty of Geosciences, Geoengineering and Mining  
of the Technische Universität Bergakademie Freiberg

approved

## Thesis

to attain the academic degree of

Doctor rerum naturalium  
(Dr. rer. nat.)

submitted by **MSc. Le, Hai Ha**

born on the 3. February 1973 in Hanoi, Vietnam

Reviewers: Prof. Dr. rer. nat. habil. Helmut Schaeben, Freiberg  
Prof. Dr. rer. nat. habil. Heinrich Jasper, Freiberg

Date of the award: 20. October 2014



# Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

- Prof. Dr. Helmut Schaeben
- Prof. Dr. Heinrich Jasper
- Dr. Ines Görz
- MSc. Jan Gietzel
- MSc. Paul Gabriel

Weitere Personen waren an der Abfassung der vorliegenden Arbeit nicht beteiligt.

Die Hilfe eines Promotionsberaters habe ich nicht in Anspruch genommen. Weitere Personen haben von mir keine geldwerten Leistungen für Arbeiten erhalten, die nicht als solche kenntlich gemacht worden sind. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

MSc. Le, Hai Ha

# Declaration

I hereby declare that I completed this work without any improper help from a third party and without using any aids other than those cited. All ideas derived directly or indirectly from other sources are identified as such.

In the selection and use of materials and in the writing of the manuscript I received support from the following persons:

- Prof. Dr. Helmut Schaeben
- Prof. Dr. Heinrich Jasper
- Dr. Ines Görz
- MSc. Jan Gietzel
- MSc. Paul Gabriel

Persons other than those above did not contribute to the writing of this thesis.

I did not seek the help of a professional doctorate-consultant. Only those persons identified as having done so received any financial payment from me for any work done for me. This thesis has not previously been published in the same or a similar form in Germany or abroad.

20. October 2014

MSc. Le, Hai Ha



# Acknowledgments

I would like to express my deepest appreciation to my first supervisor Professor Helmut Schaeben for his guidance, advice, and immense support throughout my PhD research studies. He provided the initial ideas for this thesis and encouraged and helped me overcome difficulties when performing this study. Without his help, I would not have been able to finish this thesis.

I would also like to thank my co-supervisor Professor Heinrich Jasper for his constant support and kind advice over the years. He often gave me encouraging comments when reading my manuscripts.

To my friends, I want to thank them for all of their help and discussion. Thanks to Chi Ngoc Le, Ramadan Abdelaziz, and Minh Phuong Kieu for their language correction in my manuscripts.

I would like to express my thanks to my colleagues at the Institute of Geophysics and Geoinformatics for providing an excellent atmosphere. Thanks to Dr. Ralf Hielscher and Dipl. Geol. Peggy Hielscher for reading and providing comments on my first manuscript. Thanks to Dr. Ines Görz for providing instruction on gOcad, for providing sample data, and for working as my co-author on our paper. I would also especially like to thank Jan Gietzel and Paul Gabriel for giving me the opportunity to work with the GST system.

To Technische Universität Bergakademie Freiberg, I would like to express my thanks for providing excellent material, logistical and human conditions. Thanks to the Centre of Advanced Study and Research, where I obtained useful short courses and met great friends, and I would like to thank the International Centre, where I obtained language courses.

I would like to express my thanks to my government for the financial support through the Vietnam International Education Development (VIED), Ministry of Education and Training (MOET). I would also like to express my thanks to the “Deutscher Akademischer Austausch Dienst – DAAD” for their assistance and the additional financial contribution.

Last, but not least, I would like to thank my family, my parents, my wife, and my three children, for their love, support, and encouragement and for inspiring me to follow my dreams.



## Abstract

The development of spatio-temporal geoscience information systems (TGSIS) as the next generation of geographic information systems (GIS) and geoscience information systems (GSIS) was investigated with respect to the following four aspects: concepts, data models, software, and applications. These systems are capable of capturing, storing, managing, and querying data of geo-objects subject to dynamic processes, thereby causing the evolution of their geometry, topology and geoscience properties. In this study, five data models were proposed. The first data model represents static geo-objects whose geometries are in the 3-dimensional space. The second and third data models represent geological surfaces evolving in a discrete and continuous manner, respectively. The fourth data model is a general model that represents geo-objects whose geometries are  $n$ -dimensional embedding in the  $m$ -dimensional space  $R^m$ ,  $m \geq 3$ . The topology and the properties of these geo-objects are also represented in the data model. In this model, time is represented as one dimension (valid time). Moreover, the valid time is an independent variable, whereas geometry, topology, and the properties are dependent (on time) variables. The fifth data model represents multiple indexed geoscience data in which time and other non-spatial dimensions are interpreted as larger spatial dimensions.

To capture data in space and time, morphological interpolation methods were reviewed, and a new morphological interpolation method was proposed to model geological surfaces evolving continuously in a time interval. This algorithm is based on parameterisation techniques to locate the cross-reference and then compute the trajectories complying with geometrical constraints. In addition, the long transaction feature was studied, and the data schema, functions, triggers, and views were proposed to implement the long transaction feature and the database versioning in PostgreSQL. To implement database versioning tailored to geoscience applications, an algorithm comparing two triangulated meshes was also proposed. Therefore, TGSIS enable geologists to manage different versions of geoscience data for different geological paradigms, data, and authors.

Finally, a prototype software system was built. This system uses the client/server architecture in which the server side uses the PostgreSQL database management system and the client side uses the gOcad geomodeling system. The system was also applied to certain sample applications.



# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem statement . . . . .	1
1.2	Objectives of the thesis . . . . .	4
1.3	Methodology . . . . .	4
1.4	Major contributions of the thesis . . . . .	6
1.5	Structure of the dissertation . . . . .	7
<b>2</b>	<b>Related studies</b>	<b>9</b>
2.1	Two- and three-dimensional GIS . . . . .	10
2.2	3D spatial databases . . . . .	12
2.3	Temporal GIS . . . . .	17
2.4	Geomodeling . . . . .	20
2.4.1	Implicit geological modelling . . . . .	22
2.4.2	3D surfaces . . . . .	23
2.4.3	Automatic building of structured geological models . . . . .	25
2.4.4	gOcad: a geomodeling system . . . . .	26
2.4.5	Discrete smooth interpolation (DSI) . . . . .	27
2.5	Surface representations . . . . .	29
2.5.1	Mathematical surface representations . . . . .	29
2.5.2	The surface approximation . . . . .	29
2.5.3	Conversions between representations . . . . .	30
2.5.4	Triangulated surface data structures . . . . .	32
<b>3</b>	<b>Data Models</b>	<b>35</b>
3.1	A 3D spatial data model . . . . .	36

3.2	A data model for multi-instance surfaces . . . . .	39
3.3	A data model for continuously evolving surfaces . . . . .	40
3.4	The TGSIS data model . . . . .	43
3.4.1	Geometric modeling based on topology – generalised maps	43
3.4.2	Objects in spatio-temporal domain . . . . .	48
3.4.3	Geometric model in spatio-temporal domain . . . . .	52
3.4.4	The model in object-relational form . . . . .	54
3.4.5	Assigning geoscience properties to geometry . . . . .	54
3.5	A model for multiple indexed geoscience data . . . . .	57
3.6	Summary . . . . .	59
<b>4</b>	<b>Morphological Interpolation</b>	<b>61</b>
4.1	Introduction . . . . .	61
4.2	Interpolation using mathematical morphology . . . . .	63
4.2.1	Mathematical morphology . . . . .	63
4.2.2	Morphological interpolation in the framework of.. . . .	64
4.3	Interpolation using mesh parameterisation . . . . .	69
4.3.1	Mesh parameterisation . . . . .	69
4.3.2	Cross-parameterisation and compatible remeshing . . . . .	76
4.4	TGSIS morphological interpolation . . . . .	78
4.4.1	Cutting . . . . .	80
4.4.2	Setting up Constraints . . . . .	81
4.4.3	Partition . . . . .	82
4.4.4	Calculating . . . . .	84
4.5	Summary . . . . .	89
<b>5</b>	<b>Long Transaction</b>	<b>91</b>
5.1	Introduction . . . . .	91
5.1.1	Short and long transactions . . . . .	91
5.1.2	The benefit of long transactions for TGSIS . . . . .	92
5.2	Terminology . . . . .	95
5.3	ArcGIS/ArcSDE versioning . . . . .	98
5.4	TGSIS database versioning and Oracle Workspace Manager . . . . .	101
5.4.1	Data structure and functions . . . . .	102
5.4.2	Tailoring for geological surfaces . . . . .	107
5.5	TGSIS mesh comparison . . . . .	107

---

5.6	Version merging strategy . . . . .	110
5.7	Summary . . . . .	111
<b>6</b>	<b>Prototype software</b>	<b>113</b>
6.1	Introduction . . . . .	113
6.2	Architecture . . . . .	114
6.3	Server side – PostgreSQL . . . . .	115
6.4	TGSIS Manager . . . . .	115
6.5	TGSIS gOcad plugin . . . . .	118
<b>7</b>	<b>Sample applications</b>	<b>123</b>
7.1	Introduction . . . . .	123
7.2	Modeling geological processes . . . . .	123
7.2.1	Evolution of the geometry of a salt–dome . . . . .	123
7.2.2	Geometrical modeling of syn–sedimentary faulting . . . . .	124
7.2.3	Modeling deposition and erosion of river sedimentary rocks . . . . .	126
7.3	Modeling a geological structure with certain versions . . . . .	128
7.4	Managing frequently updated models . . . . .	131
<b>8</b>	<b>Conclusions and Recommendations</b>	<b>135</b>
8.1	Conclusions . . . . .	135
8.2	Recommendations . . . . .	137





# List of Figures

---

1.1	A mind map of this thesis . . . . .	5
2.1	Examples of a raster DEM (a), a TIN (b) . . . . .	11
2.2	Geometric classes in SFA . . . . .	13
2.3	Schema for feature tables using predefined data types . . . . .	14
2.4	Schema for feature tables using SQL with Geometry Types . . . . .	15
2.5	A subsurface workflow supported by RESQML . . . . .	16
2.6	Conceptual model for 3D SDO_GEOMETRY . . . . .	17
2.7	The data model for Raza’s spatiotemporal data type . . . . .	18
2.8	The primary steps of a geological characterisation process . . . . .	21
2.9	A structural model consisting of faults and horizons . . . . .	23
2.10	A 3D volume model (a) and a 3D irregular grid (b) . . . . .	23
2.11	The relationships between two intersecting surfaces (a) on lap... .	25
2.12	Edges in a GES . . . . .	26
2.13	Splitting the domain into smaller segments . . . . .	30
2.14	An example of indexed-face-set data structure . . . . .	32
2.15	Face-based with connectivity information data structure . . . . .	33
2.16	Winged-edge data structure . . . . .	34
2.17	Halfedge data structure . . . . .	34
3.1	Simplices and geometry objects . . . . .	36
3.2	Object-relational data schema of the geometry objects. . . . .	37
3.3	An example of attaching the porosity properties ... . . . .	38
3.4	Data schema for the properties . . . . .	39
3.5	A graphic representation of Equation 3.1 . . . . .	40

[illegible]

5.9	ArcSDE as a gateway to DBMSs . . . . .	98
5.10	ArcSDE states and the state tree . . . . .	98
5.11	ArcSDE version tree . . . . .	99
5.12	Versions and states in an editing session . . . . .	99
5.13	An example of the database after a system crash . . . . .	100
5.14	ArcSDE versioning data schema . . . . .	101
5.15	An ArcSDE versioned view . . . . .	102
5.16	Version and revision . . . . .	102
5.17	Core data schema . . . . .	104
5.18	Surface schemata . . . . .	107
5.19	Mesh comparison . . . . .	109
5.20	An example of k-d tree sliding–median–widest–spread . . . . .	110
6.1	The architecture of TGSIS . . . . .	114
6.2	The graphical user interface of TGSIS Manager . . . . .	115
6.3	The list of features in a feature class . . . . .	116
6.4	Menus of the TGSIS Manager . . . . .	116
6.5	Dialog for editing features . . . . .	117
6.6	Find features and their existing time instances... . . . .	117
6.7	Find features and intervals or time points . . . . .	118
6.8	The main graphical user interface of the TGSIS gOcad plugin . .	119
6.9	The GUI of the feature management function . . . . .	119
6.10	Saving a surface into the database . . . . .	120
6.11	Loading a surface from the database . . . . .	120
6.12	The GUI of the temporal surface construction . . . . .	121
6.13	The GUI of the saving data function . . . . .	121
6.14	The GUI of the loading data function . . . . .	121
6.15	The GUI of the function to visualise data from the database . . .	122
6.16	The GUI of the function to visialise loaded temporal surfaces . . .	122
7.1	A salt dome in time. (a) $t = 1$ , (b) $t = 2/3$ , (c) $t = 1/2$ , (d) $t = 0$	124
7.2	Given data at time instances. . . . .	125
7.3	The geological horizon $B$ in time interval $[2, 3]$ . . . . .	126
7.4	The geological structure of an area of interest at time $t = 2.50$ . .	126
7.5	Given data at six time instances . . . . .	127
7.6	Top surface $B$ in time interval $[1, 2]$ . . . . .	128

7.7	The geological structure of an area of interest at time $t = 1.5$ . . .	128
7.8	Three versions of one stratigraphic horizon... . . . . .	130
7.9	A linear sequence of versions . . . . .	132
7.10	Database schema of the sample . . . . .	132

# List of Tables

---

5.1	Sample data from the VERSIONEDTABLE table . . . . .	105
-----	---	-----



## Acronyms

0D	Zero dimensional
1D	One dimensional
2D	Two dimensional
3D	Three dimensional
4D	Four dimensional
BRep	Boundary Representation
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
DEM	Digital Elevation Model
GIS	Geographic Information System
GIScience	Geographic Information Science
GML	Geography Markup Language
GSIS	Geoscientific Information System
GST	Geosciences in Space and Time
IFOB	Infrastructure objects
MONET	Moving Objects in NETworks
MOST	Data model for current and future movement
OGC	Open Geospatial Consortium
RESQML	Reservoir Characterization XML Standard
SFA	OpenGIS® Simple Feature Access
TEN	TEtrahedral Network
TGSIS	Spatio-Temporal Geoscience Information System
TIN	Triangulated Irregular Network
WKB	Well Known Binary
WKBwT	Well Known Binary with Time
WKT	Well Known Text
WKTwT	Well Known Text with Time
XML	eXtensible Markup Language





# Chapter 1:

## Introduction

---

### 1.1 Problem statement

Beginning with computer mapping software in the 1970's, the current geographic information systems (GIS) have enormously advanced and have been widely used in nearly all fields, including science, government, business, and industry. GIS are systems not only used for capturing, storing, manipulating, analysing, managing, and presenting all types of geographical data but also for integrating geographic information into data warehouse as an integrated part of enterprise decision support systems. Although GIS are primarily considered a technology, the science behind them is defined by the term geographic information science (GIScience). GIScience is a field of study that seeks to redefine geographic concepts and their use in the context of geographic information systems (Goodchild, 2010). Currently, GIScience has reached a mature level, with its own journals, research institutes, professional organisations, and conferences (Wright, 2010). However, in GIScience/GIS, many issues require further research. With respect to data managed by GIS, two primary research directions that have been considered are the following: (1) “true” 3-dimensional data (Abdul-Rahman and Pilouk, 2008, Breunig and Zlatanova, 2011, Gabriel et al., 2012, GiGa-Infosystems, 2014, Ravada et al., 2009, Zlatanova, 2000) and (2) temporal data (Erwig et al., 1999, Forlizzi et al., 2000, Güting and Schneider, 2005, Qi and Schneider, 2012, Raza, 2012, Schneider, 2009, Sistla et al., 1997, 1998, Xu and Güting, 2013). The first research direction also includes studies on the methodologies and standards of 3D

spatial databases (OGC, 2010, 2011, Oracle, 2013c, PostGIS, 2014). Currently, studies in the second research direction are aimed at integrating time with 2-dimensional spatial data only. With regard to data representations, the primary research goal is to integrate field-based models and object-based models (Couclelis, 1992, Goodchild et al., 2007, Kjenstad, 2006, Voudouris, 2008, 2011, Yuan, 2001).

Over more than two decades, geoscientists have found that subsurface characterisation did not simply extend the traditional GIS methods. Kelk (1991) defined the requirements for subsurface characterisation and modelling as follows: “The industry requires a system for interactive creation of spatial and spatio-temporal models of the physical nature of portions of the Earth’s crust, i.e., the capability to effectively model and visualise: the geometry of rock- and time-stratigraphic units, the spatial and temporal relationships between geo-objects, the variation in internal composition of geo-objects, the displacements or distortions by tectonic forces, and the fluid flow through rock units”. A series of sophisticated 3-dimensional modelling technologies that are collectively identified as geoscientific information systems (GSIS) have been developed to address these requirements (Turner, 2000, 1991, 2006, Turner and Gable, 2007). The term “geoscientific information systems” and its acronym GSIS are preferred because they offer a degree of parallelism to the widely adopted term “geographic information systems” and the acronym GIS. Similar to the evolution of GIS, the current GSIS are in the first phase of their evolution, i.e., computer 3-dimensional mapping, and therefore sometime considered as “geomodeling”. The next step in the evolution of GSIS is to develop primary functions, such as those in GIS. These functions include storing data in a data centre (database system), querying, analysing, and representing.

To gain insights into geographical and geological processes, the time factor must be considered. Therefore, many studies have been aimed at integrating time into geographical and geological data since the 1990s (Peuquet, 1994, Worboys, 1994). More recently, studies have been established to study GIS/GSIS in larger dimensions including time (Caumon, 2010, Ohori et al., 2013b, van Oosterom and Stoter, 2010). In these studies, the fourth dimension can be interpreted as time, the fifth dimension can be interpreted as scale or uncertainty, and certain larger dimensions can be interpreted as any non spatial dimensions. In our study, multi-dimensional time can be represented by these larger dimensions. Unfortunately,

to our knowledge, the statement “In current systems, the temporal dimension often plays a subordinate role, temporal variation being represented by a series of static snapshots” (Worboys, 1994) remains true.

The goal of this thesis was to study and contribute to the development of the next generation of GIS in which time dimensions are considered. As indicated in the title of this thesis, “Spatio-temporal Information System for the Geosciences: Concepts, Data models, Software, and Applications”, our approach includes studying concepts, proposing data models, building prototype software, and testing software with certain sample applications. We use the term “spatio-temporal geoscience information systems” (or TGIS) instead of the GIS term to emphasise that these systems consider time to be an integrated component of the data. By integrating space and time, TGIS are simpler to process and answer many queries, such as “Given a location (geo-object at a specified location) and properties, such as temperature and pressure, for what period(s) of time was the geo-object exposed to the location and the properties?” or “What are the geometry and properties of a geo-object at a given time?”.

The primary issues raised by TGIS are as follows. The first issue is how to manage objects with complex geometries to achieve fast access and processing. Because a complex geometric object is always partitioned and approximated by a set of components, the term “micro topology” (hereafter referred to as “topology”) is introduced to refer to the relationships between the components. Moreover, the topology should be stored explicitly to achieve fast access and processing of the object. The second issue is related to the physical properties of the geological objects. These properties are surveyed or observed at certain locations in the object. For other places in the object, the properties are estimated, typically using interpolation algorithms. It is necessary to attach the properties to each geometric component of the object. These properties can then be used to query the object or its components or to run numerical simulations. The third issue involves integrating the time dimension with space. Many questions lead to the requirement of considering space to be a discrete or continuous function of time. The fourth issue is data modeling. From the surveyed or interpreted data representing an object at certain time points, the question is how to model data that are representative of the evolution of the object. In addition, operations, such as model updating in the geosciences may be time intensive and may span multiple working sessions. Moreover, certain historical states of the data require

persistent storage. Therefore, the fifth issue is related to database versioning and long transactions.

## 1.2 Objectives of the thesis

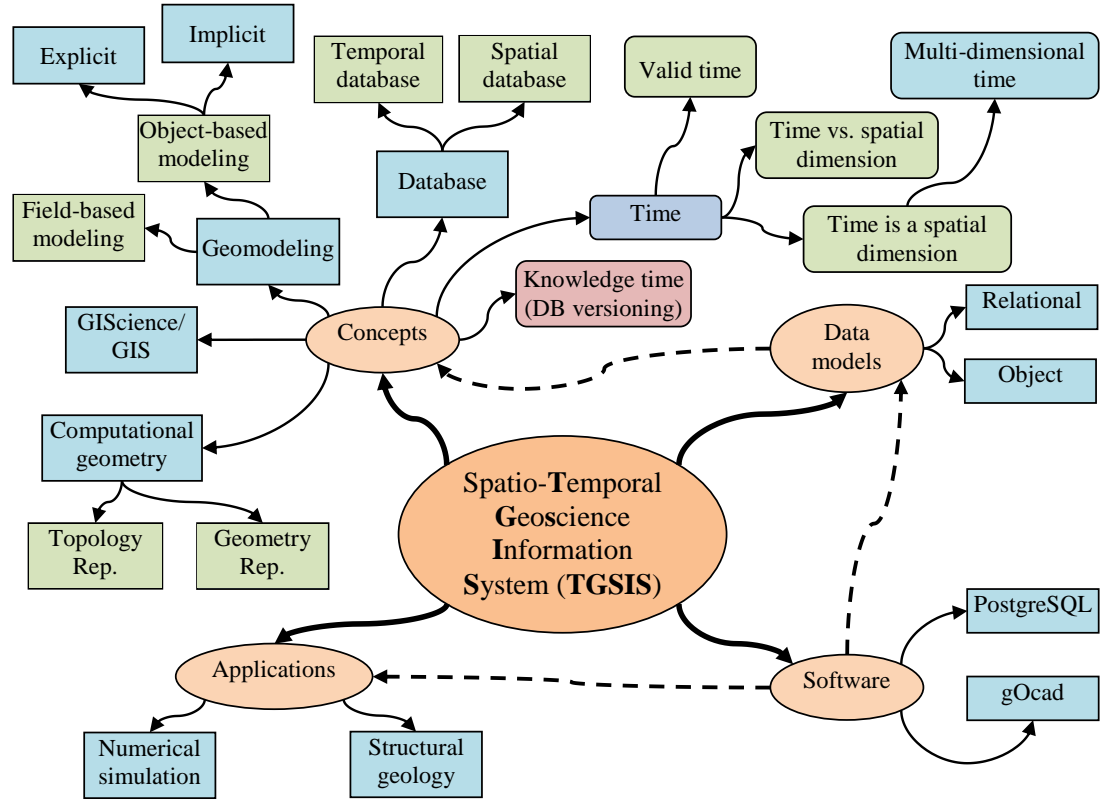
The goal of this study is to contribute to the development of TGSIS by studying and proposing solutions for the previously mentioned issues. The objectives of this study are as follows:

- Study and analyse the 2-dimensional and 3-dimensional geographic information science/systems (GIScience/GIS), standards and methodologies of 3D spatial databases, temporal GIS, geomodeling, and surface representations.
- Propose data models (database schemata) which represent geo-objects with the geometry, topology, and properties. These data models are required to represent the integration of time and space in which, sometime, space is considered as a discrete or continuous function of time.
- Study and propose methods for modeling temporal data that are representative of the evolution of an object.
- Study and propose solutions to archive persistently certain historical states of the data and different data versions in a database.
- Build a prototype software operating with primary GIS functions, such as data capturing, manipulating, and querying.
- Test the software with certain sample applications.

## 1.3 Methodology

Because the TGSIS have their roots in the GIScience/GIS and geomodeling, the concepts and requirements of TGSIS can be obtained from studies on these systems. Primary GIS functionalities, such as data capturing, storing, manipulating, querying, managing, analysing, and representing, are required in TGSIS. The requirements and difficulties of a geomodeling system must be carefully studied. Two primary approaches, field-based and object-based, are currently used to

represent geological phenomena. Moreover, explicit and implicit modelling are the two methods used to model a geo-object. A mind map of the thesis is shown in Figure 1.1.



**Figure 1.1:** A mind map of this thesis

Many of the advantages of spatial data modeling that have been developed within computational geometry, computer graphics, computer vision, CAD, and CAM, must be applied to TGSIS to create and manipulate complex geometries in the geosciences. Additionally, as GIScience/GIS do, TGSIS must consider the advantages of spatial and temporal database technologies. Furthermore, considering as one or multiple dimensions, time is an essential component of TGSIS. There are certain arguments that the time dimension is significantly different from spatial dimensions; therefore, time must be distinguished from spatial dimensions. In this approach, the geometry, topology, and other properties of a geo-object should be represented as functions of time (Peuquet, 1994). In contrast, time can be considered as one or more spatial dimensions; therefore, multi-dimensional time and space can be represented using a larger spatial dimensional representation (Ohori et al., 2013b, Worboys, 1994). In addition to

time, other dimensions, such as uncertainty, scale, and version, can be considered in TGSIS. Using database versioning, TGSIS can manage many different versions of geoscience data for different geological paradigms, source data, and authors.

Based on the concepts and requirements of TGSIS and the methodologies of object/relational database design, TGSIS data models are proposed. Applications such as modelling and numerical simulations in structural geology and groundwater studies have been reviewed. Finally, a proposed software based on gOcad and PostgreSQL is built and tested.

## 1.4 Major contributions of the thesis

This thesis concentrates on information systems that address geometry data in the 3-dimensional space or the multi-dimensional space, the topology and geoscience properties that discretely or continuously change over time. The primary contributions are as follows:

- A review of geographic information geoscience/systems (GIScience/GIS), 3-dimensional spatial databases, and temporal GIS is performed.
- The concepts and requirements of geomodeling, and geomodeling processes are studied.
- Surface representations are reviewed.
- Data models (database schemata) for geoscience data referenced to space and time are proposed.
- The morphological interpolation theory is presented.
- A new morphological interpolation method based on parameterisation techniques and tailored for geological surfaces is proposed to model temporal data.
- The long transaction concept is described. A review of the implementations of long transactions in the ArcGIS/ArcSDE and in the Oracle Workspace Manager is also presented.
- A new data schema with functions, triggers, and views for the implementation of long transactions and database versioning is proposed.
- The prototype software was built and tested.

- Certain sample applications as examples of the software were performed.
- Three papers have been published, namely:
  - [1] Le, H.H., Gabriel, P., Gietzel, J., Schaeben, H., **2013**. An Object-Relational Spatio-Temporal Data Model. *The journal of Computers & Geosciences* 57, 104-115. doi: 10.1016/j.cageo.2013.04.014;
  - [2] Le, H.H., **2013**. Spatio-Temporal Data Construction. *ISPRS International Journal of Geo-Information* 2(3), 837-853. doi: 10.3390/ijgi2030837;
  - [3] Le, H.H., Schaeben, H., Jasper, H., Görz, I., **2014**. Database versioning and its implementation in Geoscience Information Systems. *The journal of Computers & Geosciences* 70, 44–54. doi: 10.1016/j.cageo.2014.05.011.

## 1.5 Structure of the dissertation

This dissertation documents the results of our work. The dissertation is subdivided into eight chapters as follows:

**The first chapter** (current chapter 1) states the problem, presents the objective and methodology, and lists the primary contributions of this study.

**The second chapter** (2) reviews related studies and presents the concepts, methodologies and standards in 3-dimensional GIScience/GIS, 3D spatial databases, temporal GIS, geomodeling systems, and surface representations.

**The third chapter** (3) presents five proposed data models in their object-relational forms. The data models represent geological objects that exist in the 3-dimensional space and evolve over time. The data models consider the geometry, topology, and physical properties of the geological objects.

**The fourth chapter** (4) concentrates on studying the basis theory of morphological interpolation methods. Certain methods are based on the mathematical morphology theory, and operate on multi-dimensional binary images, which is one type of the implicit representation. A method based on parameterisation techniques and tailored for geological surfaces is proposed in the scope of this study.

**The fifth chapter** (5) describes long transactions and the database versioning in theory and technique. The proposed method, called TGSIS database versioning, is reviewed in this chapter.

**The sixth chapter** (6) describes the prototype software with the server-side module implemented in the PostgreSQL, the client-side module for constructing, visualising, and querying data implemented as a gOcad plugin, and the client-side module for management implemented as stand-alone software.

**The seventh chapter** (7) describes certain sample applications, including a simulation of tectonic, faulting, deposition, and erosion processes, and a sample geological structure model.

**The eighth chapter** (8) is devoted to the conclusions and recommendations. The primary contributions are summarised, and recommendations are given.



## Chapter 2:

# Related studies

---

As indicated by the name of our study, “spatio-temporal geoscience information systems (TGSIS)”, this research is rooted in geographic information science/systems (GIScience/GIS), spatial databases, temporal GIS, geomodeling systems, and surface representations. For the purpose of brevity, the term “2D object” is referred to as an object whose geometry is 0-, 1-, or 2-dimensional embedding in 2-dimensional space; the term “3D object” is referred to as an object whose geometry is 0-, 1-, 2-, or 3-dimensional embedding in 3-dimensional space; the term “3D surface” is referred to as a 2-dimensional manifold (surface).

Geographic information systems have become a popular tool in many fields of science, government, business, and industry. GIS were originally developed to manipulate 2D objects in maps. Currently, there are certain extensions that enable GIS to store 3D objects. However, many queries for these objects are not simple to answer. Spatial databases were inspired by GIS, but their success extends beyond the original goal. The spatial database technology supports many spatial data types and in turn motivates the development of 3D GIS. Currently, efforts have been made to develop temporal GIS integrating time into GIS to manage the evolution of 2D objects. In these systems, temporal objects are typically represented as the spatial objects existing or continually changing over a certain period. Geomodeling systems provide methodologies and techniques to model geological objects from sparse observed/surveyed data. A geological model represents geological objects in the truly 3D space with the topologies and physical properties. Surface representations are grouped together into the follow-

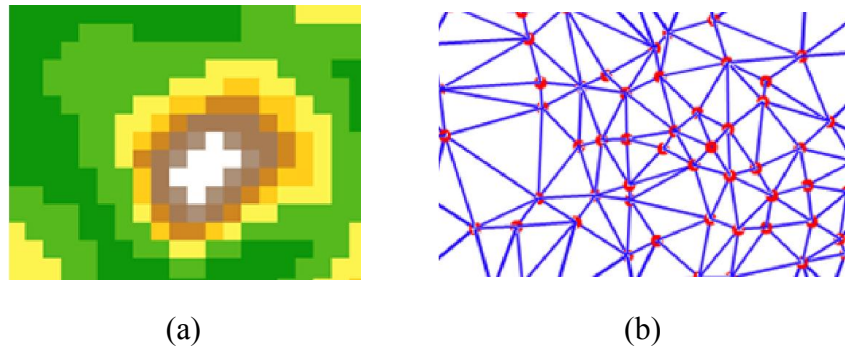
ing two primary categories: parametric and implicit representations. A surface representation concerns the data structure, the validating rules, and the manipulating operations. TGSIS are based on the systems and methodologies mentioned above. TGSIS are aimed at managing geological objects in the integrated space of 3-dimensional space and time, and they must provide tools, such as constructors, accessors, and query operators.

In this chapter, we investigate certain concepts, software, standards, and literature. Section 2.1 presents certain concepts in GIS, emphasising the 3-dimensional extension. Section 2.2 reviews certain standards and literature related to 3D spatial databases. Two spatial databases, Oracle and PostgreSQL/PostGIS, are reviewed in this section. Temporal GIS are reviewed in Section 2.3. Section 2.4 presents certain techniques used to build geological models. Surface representations and certain related issues are presented in Section 2.5.

## 2.1 Two- and three-dimensional GIS

Geographic information systems (GIS) are information systems that manage objects referenced in the 2D geographical space. GIS has applications in various fields, including but not limited to managing, planning, engineering, insuring, and transport/logistics. Originally, GIS were built to manage objects in the maps, i.e., 2D objects. The geometries of the objects in the GIS are represented by points, lines, and polygons. In an effort to truly represent objects that exist in 3-dimensional space (our real world), certain GIS systems have used the Digital Elevation Model (DEM) to represent terrain surfaces. A DEM can be represented as a raster (raster DEM) or as a vector-based Triangular Irregular Network (TIN). Using raster DEM, surfaces are stored in the 2D grid format with uniformly spaced cells, and height values (elevations) are attached to each cell. A TIN is constructed by triangulating a set of vertices (points). Examples of a raster DEM and a TIN are shown in Figure 2.1.

In current GIS systems, a TIN is constructed by first projecting the 3D input points onto a local planar surface and then applying the 2D Delaunay triangulation method. Therefore, raster DEM and TIN in GIS are only delimited to represent surfaces with single  $Z$  values, i.e., they cannot represent caves, overhangs, or arches.



**Figure 2.1:** Examples of a raster DEM (a) and a TIN (b) (ESRI, 2014b)

Certain studies have proposed a framework for a 3D GIS (Abdul-Rahman and Pilouk, 2008, Apel, 2004). Apel (2004) proposed a data model based on the gOcad (Paradigm-GOCAD, 2014) data model. The data model added the concept of observation points to the boundary-representation (BRep)-based data model of gOcad and is conformable with the GML specification for geospatial data exchange using XML format. The Tamino XML database management system was chosen to implement this data model. Abdul-Rahman and Pilouk (2008) proposed data structures for TIN and *TEtrahedral Networks* (TEN) using the relational and object-oriented approaches. These authors also presented supporting algorithms for these data structures, such as 3D distance transformation, 3D Voronoi tessellation, and TEN generation.

The ArcGIS product family (ESRI, 2014b) with the ArcGIS 3D Analyst module (ESRI, 2014a) and Esri CityEngine (ESRI, 2014d), provides tools for creating, visualising, and analysing GIS data in a 3-dimensional context. Three-dimensional GIS data managed by ArcGIS include feature data and surface data. The 3D feature data store  $z$ -values as part of the coordinates; therefore, they can potentially support many different  $z$ -values for each  $x, y$  location. The typical types of 3D feature data include *multipatches*, *3Dpoint*, *3Dpolyline*, and *3Dpolygon* (see (ESRI, 2014a) for descriptions of these types). The surface data include raster DEM and TIN which support only a single  $z$ -value for each  $x, y$  location. Typically, the surface data are used to represent topographical surfaces. A 3D surface is typically derived from sample point, line, or polygon data using interpolation or triangulation. Certain interpolation methods are *Inverse Distance Weighted*, *Spline*, *Kriging*, and *Natural Neighbor*. The triangulation method is a (constrained) Delaunay triangulation of the plan. Note that 3D implemented in ArcGIS is not “true” 3D in the geosciences’ point of view.

The GST-Framework developed by GiGa Infosystems (GiGa-Infosystems, 2014) has its initialization in the geoinformatics working group at the Department of Geophysics and Geoinformatics, TU Bergakademie Freiberg, which provides components for storing, managing and visualizing three dimensional geoscience data with an emphasis on the interoperability (Gabriel et al., 2012, GiGa-Infosystems, 2014). The 3D geometry model of the GST-Framework is compliant with the normalised schema of *SQL-implementation based on predefined data types*, which is defined in the OpenGIS simple feature access (OGC, 2010, 2011).

## 2.2 3D spatial databases

GIS have inspired the development of spatial databases, but the success of spatial databases is beyond the original goal. Spatial databases are used in a wide variety of real-world applications. Breunig and Zlatanova (2011) have presented a 25-year retrospective and future directions of the geo-database studies. These authors have emphasised that new 3D geo-databases are required to manage surface and volume models, and geo-databases play a central role as data integration and handling platforms for geo-referenced 2D and 3D data in applications such as 3D urban planning, environmental monitoring, infrastructure management, and early warning or disaster management and response systems.

In this study, we review two standards and two database management systems that support 3-dimensional data. The first standard is OpenGIS<sup>®</sup> simple feature access (SFA), also called ISO 19125, which specifies a common storage model for geographic data (OGC, 2010, 2011). The second standard is RESQML which is an exchange format standard for transferring earth model data between applications in a vendor neutral, open, and simple format (RESQML, 2012). The two database management systems selected are Oracle Spatial (Oracle, 2013c) and PostGIS (PostGIS, 2014).

SFA consists of the following two parts: part 1 is the common architecture, and part 2 is the SQL option. Part 1 describes the common architecture for simple feature geometry. The standardised geometric classes are shown in Figure 2.2.

The standard defines 0-, 1-, and 2-dimensional geometric objects that exist in 2-, 3-, or 4-dimensional space ( $R^2$ ,  $R^3$ , or  $R^4$ ). Geometry values in  $R^2$  have points with coordinate values for  $x$  and  $y$ . Geometry values in  $R^3$  have points with

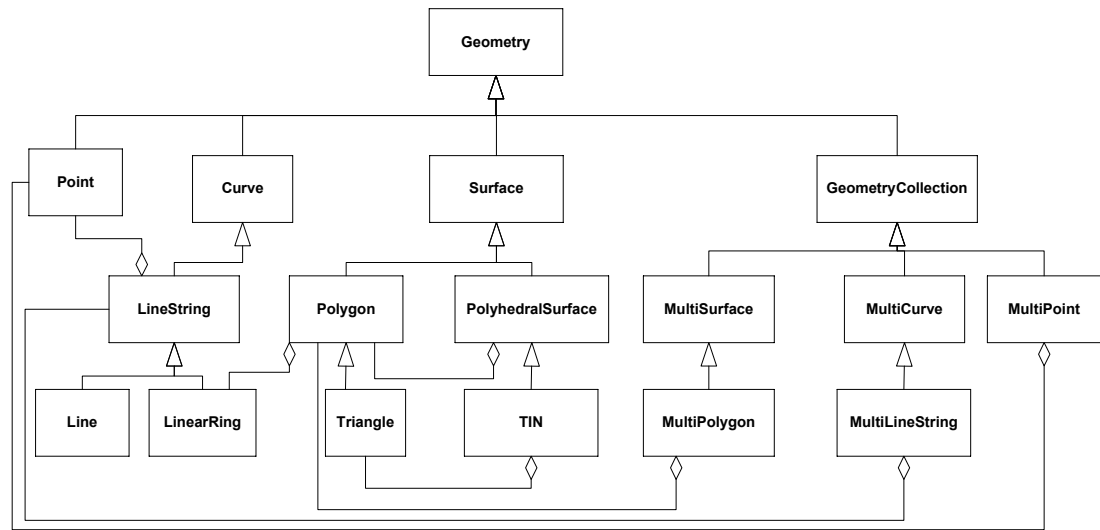


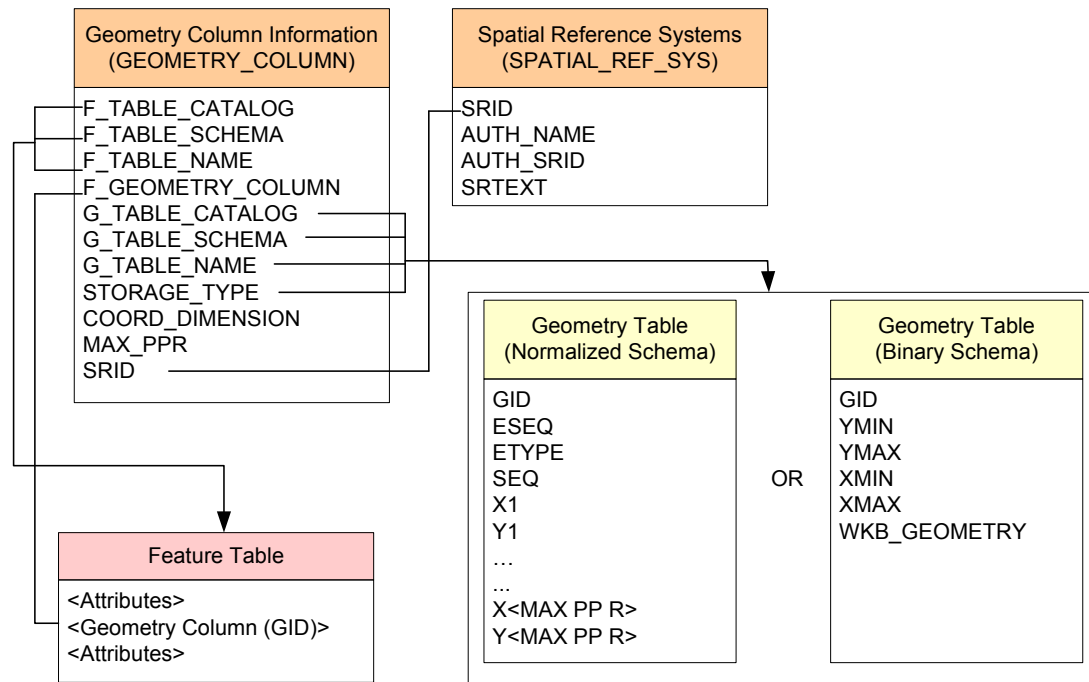
Figure 2.2: Geometric classes in SFA

coordinate values for  $x$ ,  $y$  and  $z$  or for  $x$ ,  $y$  and  $m$ . Geometry values in  $R^4$  have points with coordinate values for  $x$ ,  $y$ ,  $z$ , and  $m$ . The  $z$  coordinate of a point is typically, but not necessarily, represents altitude or elevation. The  $m$  coordinate represents a measurement. Objects are defined by a finite set of points and linear or planar interpolations between points. The geometric classes are closed under the main geometric operators, such as the boundary, intersection, union, difference, and buffer operators. The standard also defines two important formats, i.e., *well-known text representation* and *well-known binary representation*, which are the interface to the applications. Although this standard does not define solid objects, a closed *polyhedralsurface* can be used to represent them. Moreover, a polygon with four non-coplanar vertices can be considered a tetrahedron, and a *multipolygon* can be used to represent 3-dimensional tetrahedral meshes.

The second part of SFA (the SQL option) defines a standard structured query language (SQL) schema that supports storage, retrieval, query, and updating of feature collections via the SQL Call-Level Interface (SQL/CLI). The feature table schemata are presented in one of the following two SQL implementations: the implementation based on a classical SQL relational model using only predefined SQL data types, and the implementation based on SQL with additional types of geometry. In the first implementation, a geometry value is defined by certain rows in the geometry table; in the second implementation, a geometry value is defined by a value of the user-defined type (UDT). A set of SQL accessible routines

supports geometric behaviours and queries.

The standard defines a schema for the management of feature tables, geometries, and the spatial reference system information in an SQL-implementation based on the predefined data types, as shown in Figure 2.3.

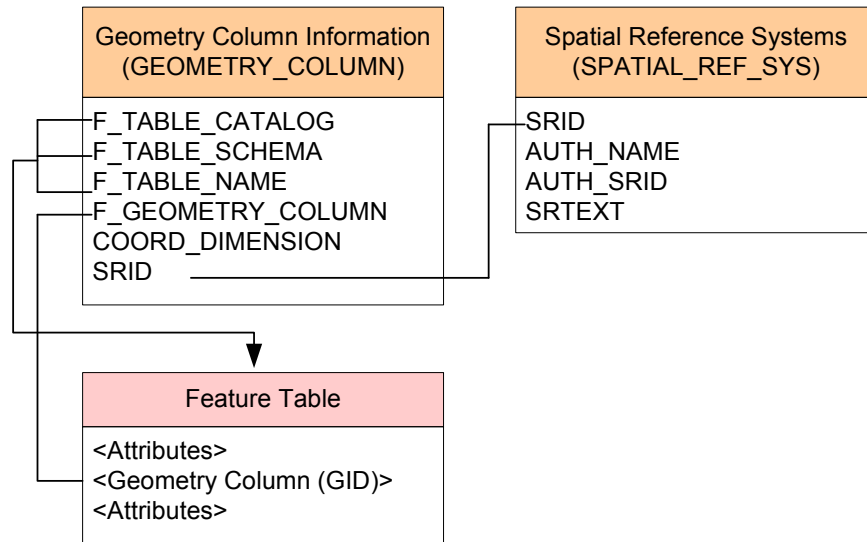


**Figure 2.3:** Schema for feature tables using predefined data types (figure redrawn from OGC (2010))

The schema in a SQL-implementation with a Geometry Type extension is shown in Figure 2.4.

Although it supports 3-dimensional space, the goal of the standard is to standardise geographic information. Nearly all of the operators are in 2-dimensional space; objects in 3-dimensional space are projected onto a horizontal surface that is typically represented on a map. The result of these operators may not be identical to that of full 3D geometry operators.

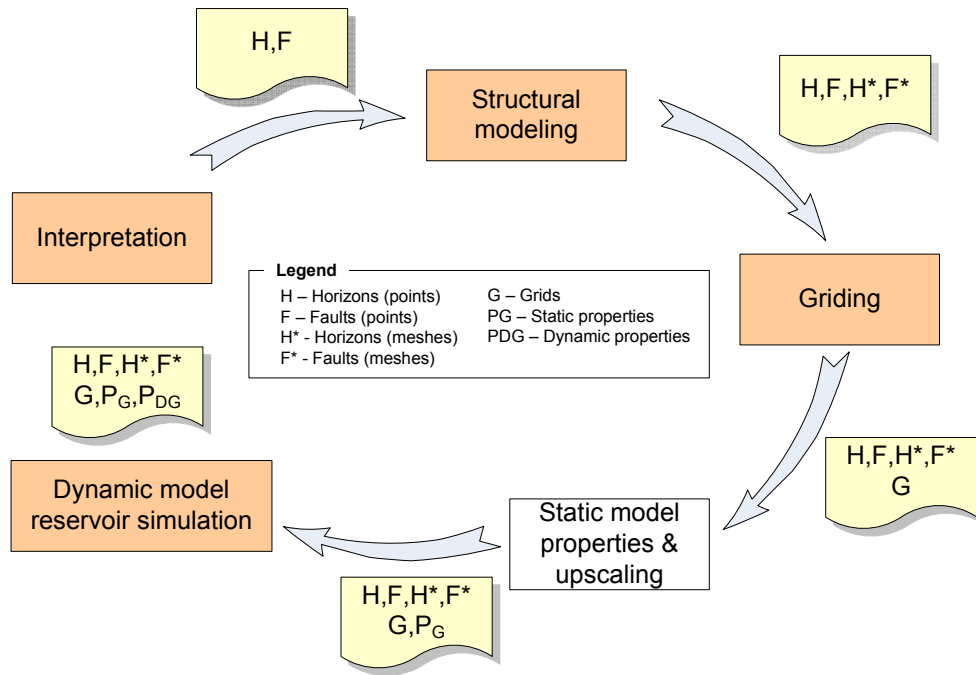
RESQML (2012) is the XML grammar defined by Energistics, a global, not-for-profit, membership organisation that was created to serve as a neutral body to facilitate and manage open data, information and process standards for the upstream oil and gas industry. The goal of RESQML is to help petro-technical professionals address data incompatibility when using the multiple software packages required for analysis, interpretation, modeling, and simulation along the en-



**Figure 2.4:** Schema for feature tables using SQL with Geometry Types (figure redrawn from OGC (2010))

tire subsurface workflow. This type of workflow is shown in Figure 2.5. RESQML organises data into horizons, faults, grids, and properties. The data have multi versions, and each version can have several geometry representations. The geometry representations for a horizon include 3D point sets, orthogonal 2D grids, triangulated meshes, and hybrids (an orthogonal grid plus triangles). The geometry representations for a fault include 3D point sets, orthogonal 2D grids, triangulated meshes, and pillar sets (a collection of 3D poly-lines). The coordinate lines of a grid are not required to be straight or monotonic functions of depth. The data in RESQML are stored in plain text format or in HDF5 format (HDF5, 2014).

Oracle Spatial is one of the database management systems that supports three-dimensional geometry objects (Oracle, 2013c). Oracle Spatial represents geometries using the object-relational model. This model is compliant with the SQL implementation with geometry types defined by the Open GIS (OGC, 2010, 2011), i.e., storing an entire geometry in the native spatial data type, SDO\_GEOMETRY. Many application models follow a workflow consisting of the following three steps: acquiring point cloud data, building surface models, and building application models. Oracle Spatial has two new 3D data types, which are SDO\_POINT\_CLOUD and SDO\_TIN (Ravada et al., 2009). A single object of the types SDO\_POINT\_CLOUD and SDO\_TIN can store up to  $4 \times 10^{18}$  points by partitioning the points into fixed-size blocks and storing the blocks as multiple rows in a separate

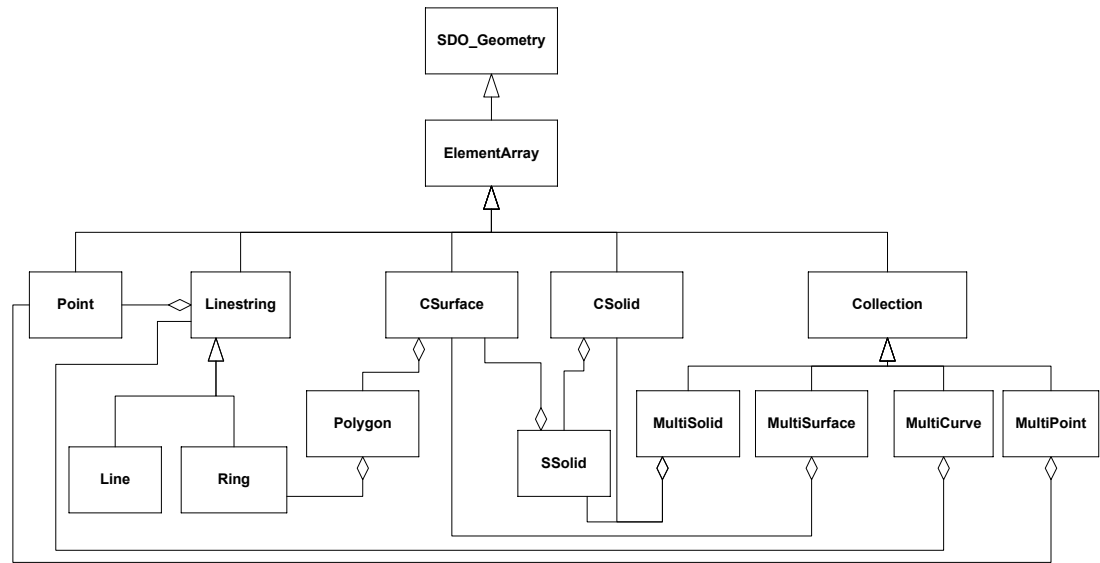


**Figure 2.5:** A subsurface workflow supported by RESQML

block table. The extent and the resolution of each block are stored to support the spatial index and multi-resolution. Figure 2.6 shows the conceptual model for SDO\_GEOMETRY. The SDO\_GEOMETRY type is an array of one or more elements, in which each element represents a point, a line-string, a surface, a solid, or a collection. A *CSurface* (composite surface) consists of one or more connected polygons. One outer surface and zero or more inner surfaces form a simple solid (*SSolid*). A *CSolid* (composite solid) is formed by a number of attached *SSolids*. The collection types consist of one or more elements of the appropriate type.

The PostgreSQL/PostGIS is a spatial database capable of handling with 3D geometries (PostGIS, 2014, PostgreSQL, 2014). The PostGIS is compliant with the OpenGIS simple feature access (OGC, 2010, 2011). The 3-dimensional geometry types in PostGIS include POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, TRIANGLE, TIN, POLYHEDRAL-SURFACE, and GEOCOLLECTION. The well-known text format and the well-known binary format with the *Z* value are primarily used for input and output data. Because the geometry backend of the current version of PostGIS is GEOS (2014), which supports 2D geometry only, PostGIS has only very basic 3D spatial functions, e.g., *ST\_3DDistance* and *ST\_3DIntersects*.





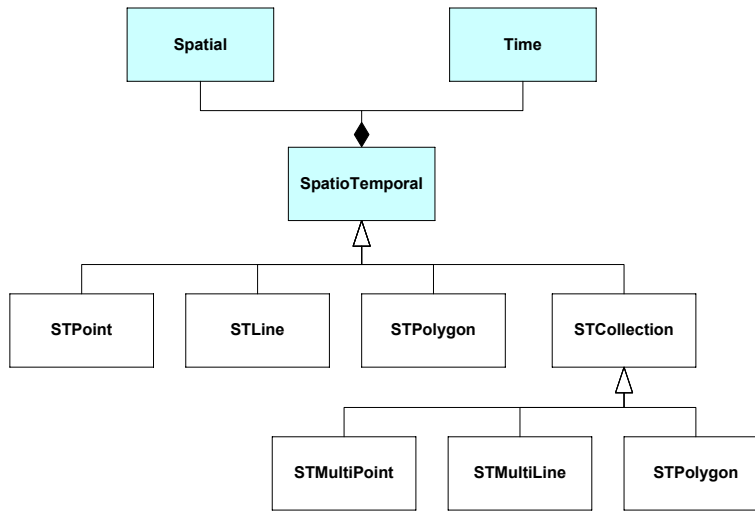
**Figure 2.6:** Conceptual model for 3D SDO\_GEOMETRY

## 2.3 Temporal GIS

Temporal GIS is one research approach used to extend GIS to support time and space (2-dimensional data). In temporal GIS, each spatial object has its life span. Using the object-oriented approach, Raza (2012) proposed a new spatiotemporal data type. The spatiotemporal class is the aggregation of the spatial and temporal classes. The spatial class represents spatial objects in 2-dimensional Euclidean space, and the temporal class represents linear time. Figure 2.7 shows the data model for the spatiotemporal data type.

The spatiotemporal class contains attributes and operations. The attributes  $ts$  (start time) and  $te$  (end time) are used to define the life span of an object of the class. The operations are further classified into the following three main categories: constructors, accessors, and operators. The constructor methods are used to build up objects. The methods are  $STT\_Type(WKTwT, SRID)$  and  $STT\_Type(WKB, SRID, T)$ , where  $WKTwT$  is an acronym for Well Known Text with Time, and  $WKB$  is an acronym for Well Known Binary. The accessor methods include  $STT\_AsText(st\_type)$ ,  $STT\_AsBinary(st\_type)$ , and  $STT\_AsShape(st\_type)$ . The operator methods continue to be divided into three categories: pure temporal, pure spatial, and spatiotemporal operators.

Other studies on temporal GIS have intended to expand the database systems



**Figure 2.7:** The data model for Raza's spatiotemporal data type

to manage moving objects (Güting and Schneider, 2005). Güting and Schneider categorised moving objects into the following two groups: moving points and moving regions. Moving points represent objects that change their positions in the plane but do not change their extents (or shapes); examples include cars, trucks, ships, and mobile phone users. Moving regions represent objects that not only change their positions but also grow and shrink, i.e., change their shapes. Hurricanes and oil spills are examples of this type of object. The term “moving objects” is used to emphasise that the geometries of the objects can continuously change. To manage moving objects in database systems, the database technology was extended using one of two strategies. The first strategy is to build a layer on top of an existing database management system and map the moving object representations and functions to the existing facilities of the database management system. The second strategy is to extend the database management system by providing new data structures, methods for querying, and algorithms for indexing and joining.

To manage mobile objects in a database, the database must be updated frequently to be able to obtain their current positions. Updates that are more frequent decrease the errors between the recorded positions and the actual positions. Conversely, less frequent updates result in larger errors. A different approach is to store a moving object not by its position directly but by its motion vector, i.e., its position as a function of time (Güting and Schneider, 2005, Sistla et al., 1997, 1998). Therefore, the updates of a motion vector are still required but are much less frequent than those in the case of storing positions. This approach

leads to a data model called the “Data model for current and future movement” or MOST. A concept in MOST is the *dynamic attribute*, i.e., an attribute whose values change over time without explicit updates. Simply, a dynamic attribute  $A$  of type  $T$  (denoted  $A : T$ ) is represented by three sub-attributes,  $A.value$ ,  $A.updatetime$ , and  $A.function$ , where  $A.value$  is of type  $T$ ,  $A.updatetime$  is of time type (e.g., real), and  $A.function$  is of function type  $f : time \rightarrow T$  such that at time  $t = 0$ ,  $f(0) = 0$ . Therefore, the value of  $A$  at time  $t$  is defined as  $value(A, t) = A.value + A.function(t - A.updatetime)$  for  $t \geq A.updatetime$ . For applications in which objects move along networks, e.g., vehicles moving along road networks, the position attribute of an object class can be modeled by an attribute type, denoted as  $loc$ , with six sub-attributes,  $loc.route$ ,  $loc.startlocation$ ,  $loc.starttime$ ,  $loc.direction$ ,  $loc.speed$ , and  $loc.uncertainty$ . Of these attributes,  $loc.route$  is a pointer to a line spatial object describing the geometry of a path over the traffic network.  $Loc.startlocation$  is the location of the moving object at time  $loc.starttime$  and is represented as a point on  $loc.route$ .  $Loc.direction$  is a Boolean function indicating the direction along the route;  $loc.speed$  is a real function of time that is used to compute the distance along the route.  $Loc.uncertainty$  can be a constant or a function of time used to represent the threshold of the deviation of the object; when the threshold is reached, the object sends a location update. The management of moving objects by storing motion vectors causes the database to only store recent and very near future information about the positions; however, no real historical data are stored in the database.

A data model and a query language for the true history of the movements have been developed based on the strategy of extending the database management system (Erwig et al., 1999, Forlizzi et al., 2000, Güting and Schneider, 2005). New data types called spatio-temporal types, such as *mpoint*, *mregion*, *mreal*, *mint*, *mstring*, *mbool*, where  $m$  represents moving, are equipped with a comprehensive collection of operations and predicates.

In addition to studies on moving objects in environments in which their positions are not impeded by any spatial constraints (unconstraint environments), many studies have considered moving objects in constrained environments. In this case, moving objects are not free to move; they are restricted by certain spatial constraints, such as spatial networks (Schneider, 2009). Qi and Schneider (2012) proposed a new two-layered data model called the MONET (Moving Objects in NETworks) model. In this model, the lower layer is a data model for

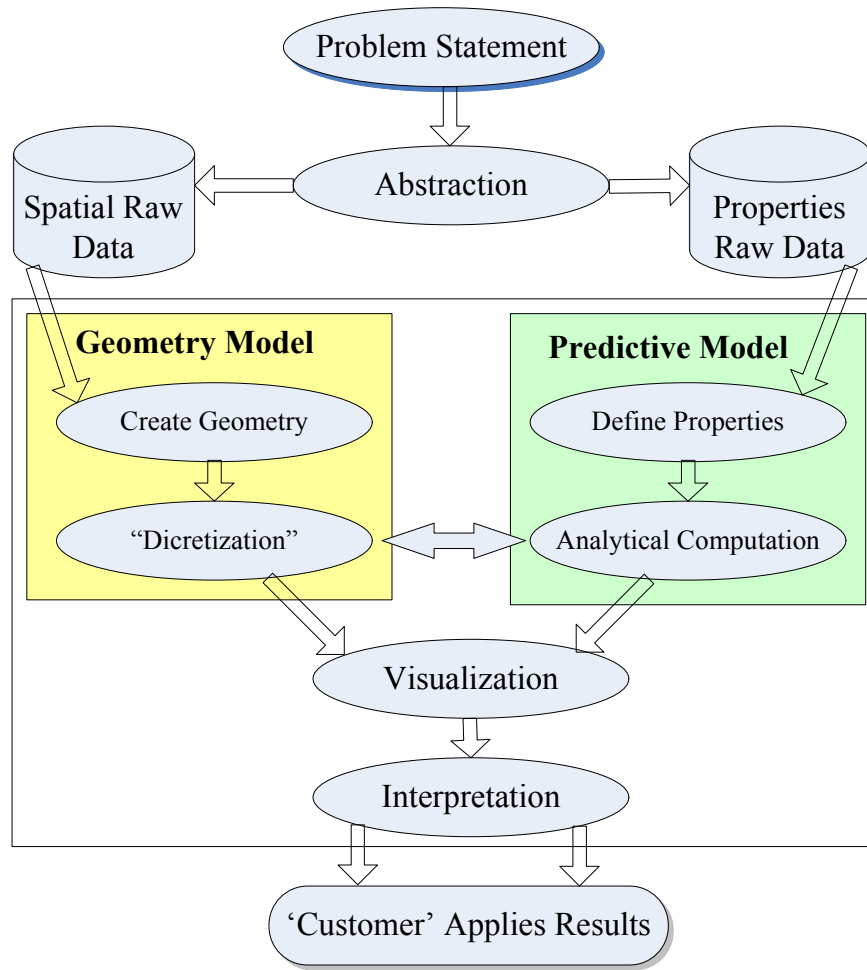
spatial networks, and the upper model is a data model for moving objects. A query language, called the MONET QL (MONET Query Language), was also proposed to access moving objects in spatial networks and to provide high-level operations on them. Xu and Güting (2013) considered moving objects that pass through several real world environments. An example of this movement is a man walking from the house to a bus stop and taking a bus to the train station, moving from one city to another by train, and finally walking from the train station to his office. The movement can be described by a sequence of transportation modes given by  $Walk \rightarrow Bus \rightarrow Train \rightarrow Walk \rightarrow Indoor$ . In this study, objects can move in five environments (called infrastructures) namely *Free Space*, *Road Network*, *Public Transportation Network*, *Region-based Outdoor*, and *Indoor*. Each infrastructure consists of a set of infrastructure objects (IFOB) and defines the available places to move. The study has modeled the available places for each environment (infrastructure) and has given data types representing its components, i.e., IFOB. This study is interesting, but it cannot be applied to GIS, at least at the present time.

## 2.4 Geomodeling

In many fields, modeling means numerical analysis and the simulation of continuous systems represented by differential equations. However, as shown in (Houlding, 1994), geomodeling does not contain differential equations but is concerned with computer techniques for geological interpretation, geostatistical prediction, and graphical visualisations of inaccessible geological conditions from limited information. The process of interpretation, prediction, and visualisation is called geological characterisation. Houlding (1994) stated that the geological characterisation must predict and interpret potentially complex geological conditions that are discrete and pseudo-continuous in terms of their spatial variability. The primary steps of a geological characterisation process were presented by Turner and Gable (2007), as shown in Figure 2.8.

Geomodeling has been defined more strictly by Mallet (2002) as in Definition 2.1.

**Definition 2.1** *Geomodeling consists of the set of all the mathematical methods allowing to model in a unified way the topology, the geometry and the physical*



**Figure 2.8:** The primary steps of a geological characterisation process (figure redrawn from (Turner and Gable, 2007))

*properties of geological objects while taking into account any type of data related to these objects.*

The goal of geomodeling is to model and understand the subsurface structures. Geomodeling involves a variety of representations describing the geometry of geological objects, their neighborhoods (topology) and the properties of their rock units. Because rock units and their groups (layers, or blocks) are solid objects, solid models are particularly interesting to geoscientists. In solid modeling, a boundary representation (BRep) defines solids by their bounding surfaces, thereby providing an efficient volume description (Caumon et al., 2004). In the explicit modelling approach, 3D surfaces are created from observed/surveyed data and then sewn up to create volumes. In contrast, the implicit modeling approach builds up volumetric functions (3D implicit functions) from surveyed data. The

iso-value surfaces of these functions can be extracted to represent the boundary of the geo-objects and sewn them with other surfaces in the boundary representations. The volumetric functions can also be used as the primary geometry in the Constructive Solid Geometry (CSG) (Natali et al., 2013).

### 2.4.1 Implicit geological modelling

Recently, the implicit modeling approach, i.e., using implicit functions (also called scalar field functions), has received much attention due to its advantages, such as noise resistance, automatic hole filling, and support of arbitrary topology. Two primary types of methods are used to build up these scalar field functions (Caumon et al., 2013). The first type uses dual kriging and radial basis function interpolation to estimate the scalar field  $f$  as described in the following equation:

$$f(x) = \sum_{l=1}^L c_l \cdot p_l(x) + \sum_{n=1}^N \lambda_n \cdot \phi(|x - x_n|) \quad (2.1)$$

where  $p_l(x)$  are polynomial basis functions,  $c_l$  are the corresponding drift coefficients,  $L$  is the total number of polynomial terms,  $N$  is the total number of data points,  $\phi(|x - x_n|)$  is either the covariance between the data point  $x_n$  and the unknown  $x$  (Chilès, 2012) or a basis function such as a thin plate spline or simply the inverse distance (also called the biharmonic function) (Carr et al., 2001), and  $\lambda_n$  are the unknown interpolation coefficients (Caumon et al., 2013). Equation 2.1 leads to a dense linear system of equations that can be quickly solved by the fast multipole method (Carr et al., 2001). Cowan et al. (2003) used this method to construct continuous and smooth geological shapes. However, discontinuities, such as faults, were not considered in this method.

The second type of method used to build up these scalar field functions uses discrete optimisation to compute the scalar field on a pre-defined volumetric mesh, e.g., a regular grid or tetrahedral mesh. Frank et al. (2007) computed this scalar field on tetrahedral meshes conforming to faults. Their method leads to a linear system of equations of  $M$  unknowns,  $f_1, \dots, f_M$ , at the mesh nodes, as follows:

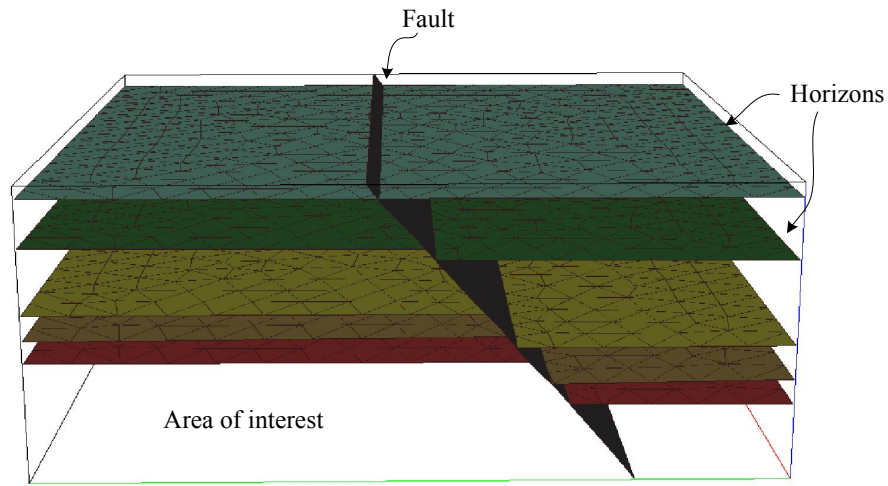
$$A \cdot [f_1, \dots, f_M]^T = [b_1, \dots, b_C]^T = b^T \quad (2.2)$$

where  $C$  is the number of linear constraints applied to the system. The coefficients of these constraints are stored in the  $C \times M$  sparse matrix  $A$  and the right-hand

side vector  $b$  (Caumon et al., 2013).

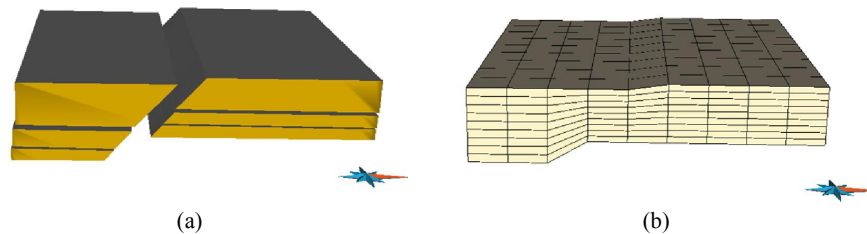
### 2.4.2 3D surfaces

In the geosciences, 3D surfaces (2-dimensional manifolds embedding in the 3-dimensional space) are often used to represent horizons, faults, unconformities, and intrusion boundaries. These surfaces model the primary discontinuities of the domain of the study. The set of these 3D surfaces makes a structural model that represents the subsurface structures. Figure 2.9 shows an example of a structural model.



**Figure 2.9:** A structural model consisting of faults and horizons

Structural models can be used to build 3D volume models for visualisation (Figure 2.10a) or can be meshed to solve geophysical and geomechanical problems. Figure 2.10b shows a 3D irregular grid built from the structural model to model physical properties.



**Figure 2.10:** A 3D volume model (a) and a 3D irregular grid (b) built from surfaces

Because it is difficult to directly access the subsurface, the data for modeling are always sparse, primarily existing along drilling paths and seismic sections.

Therefore, the modeling is not straight forward. Caumon et al. (2009) presented rules and guidelines. Using these rules, each surface should fit the available observation data within an acceptable range, depending on the type of data, data precision, and resolution. A consistent structural model does not only consist of surfaces fitting the observation data, but also the correct relationships between them. These relationships are called macro-topologies, which are used to model the borders of the objects.

Some of these rules are listed as follows (Caumon et al., 2009): (1) Geological surfaces are always orientable, i.e., have two well-defined sides (because a geological surface is a boundary between two volumes of rocks with different characteristics). (2) The rule along common a border surface intersection can be relaxed by some modeling software packages (Mallet, 2002). (3) Each boundary between two rocks must lie only on one side of one rock. (4) The only fault surfaces may have borders that are not connected to other structural model interfaces; stratigraphic surfaces necessarily terminate on faults, unconformities or model boundaries; and only faults may terminate inside rock units when the fault displacement becomes zero.

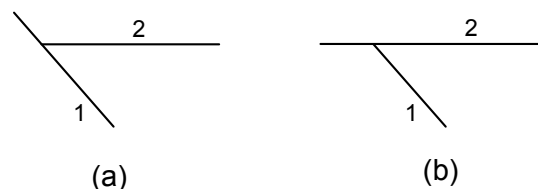
Caumon et al. (2009) indicated that fault surfaces should be built to partition the studied domain into fault blocks; stratigraphic horizons are subsequently created, following the rules described above. There are two primary methods for modeling surfaces: direct triangulation and indirect surface construction. In the direct triangulation method, triangle strips are created by directly connecting nodes of the several curves. The mesh quality can be improved by removing redundant nodes and switching triangles according to the Delaunay criterion. In the indirect surface construction method, an initial surface is created, e.g., the average plane. Next, constraints are added. Finally, an interpolation is used to minimise the data misfit. The Discrete Smooth Interpolation (DSI) used in gOcad modeling system is this type of interpolation (Mallet, 2002, Paradigm-GOCAD, 2014). In DSI, the constraints restrict the degrees of freedom of the surface nodes during the interpolation. For example, a *Control Node* constraint is used to freeze a given location; a *Straight Line* constraint allows a node to move only along a specified direction. Soft constraints are honoured in a least-squares sense by DSI, e.g., a *Control Point* constraint attracts the surface along a specific direction similar to a rubber band. In addition, DSI uses other constraints such as *Border* constraints, *Thickness and Range Thickness* constraints. *Border*



constraints are put on the borders of a surface to prevent its retraction by DSI and to kept the interaction between surfaces, e.g., horizons and fault contacts. *Thickness and Range Thickness* constraints are used to remain a distance or a range of distance between a surface and another surface.

### 2.4.3 Automatic building of structured geological models

Brandel et al. (2001, 2005) proposed a method to automatically build structural models from surfaces. The method includes two steps. In the first step, geological knowledge is recorded by a graphical language called the “Geological Evolution Scheme” (GES). The second step automatically builds the model, surface after surface, according to the instructions read from the GES. The input surfaces are assumed one of the following two types: polarised (POL) surfaces and non-polarised tectonic (TEC) surfaces. POL surfaces correspond to the limits of sedimentary formations or of intrusions; their two faces are geologically different such that one face is the older formation (F-old) and the other face is the younger formation (F-young). The TEC surfaces correspond to the following tectonic discontinuities: faults or thrust surfaces. The two faces of TEC surfaces are geologically equivalent, both facing older formations. The method also follows two fundamental assumptions that each surface, POL or TEC, has one well-determined age; therefore, when two surfaces intersect, one of them is necessarily interrupted by the other. The relationships between two intersecting POL surfaces can be one of the following two configurations: on lap or unconformity. With the on lap configuration, the older surface (on lap surface 1) interrupts the younger surface (2), and with the unconformity, the younger surface (unconformable surface 2) interrupts the older surface (1). These two configurations are shown in Figure 2.11.

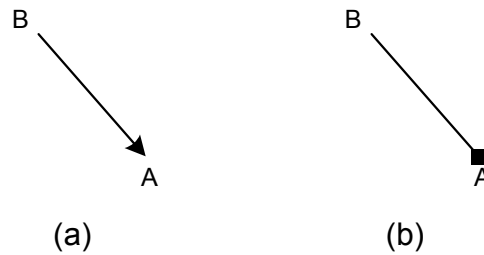


**Figure 2.11:** The relationships between two intersecting surfaces (a) on lap and (b) unconformity

To build the intersections between the surfaces as interpreted, the F-old and

F–young faces of each surface are given a value to determine the intersecting configuration.

In a GES, the nodes are placed in vertical positions that correspond to their relative ages (bottom position for old and top position for young). Two nodes can be joined by arcs of various types, as in Figure 2.12a, i.e., B follows A in the chronology, and in Figure 2.12b, i.e, fault A stops on fault B.



**Figure 2.12:** Edges in a GES

Using the notations mentioned above, a GES defines the partial or total order relationships between the surfaces of the model based on their relative ages. Because the older cannot change the younger, it is possible to automatically build the model by introducing surfaces sequentially, proceeding from the top to the bottom of the GES, i.e., from young to old.

#### 2.4.4 gOcad: a geomodeling system

In gOcad (Brandel et al., 2005, gOcad, 2014, Mallet, 2002, Paradigm-GOCAD, 2014), a geo-object is represented by its geometry, topology, and geological properties. A geo-object can be modeled by one or more gOcad objects such as *PointsSet*, *Curve*, *Surface*, *Voxet*, *SGrid*, *Well*, *Solid*, *2D-Grid*, and *Channel*. The primary component of a gOcad object is the *node* or *atom*, which refers to the  $x$ ,  $y$ , and  $z$  coordinates (for geometry), to other nodes (for topology), and to geological properties, except for the *SGrid* with centre properties. The  $x$ ,  $y$ , and  $z$  can be considered other geological properties and can be used in formulas in computation functions.

gOcad represents objects by subdividing (or tessellation) them and uses a discrete model and the Boundary Representation model. The most important algorithm is the Discrete Smooth Interpolation (DSI). From an end-user's viewpoint, the two parameters of the DSI are the smoothing and fitting factors.

Understanding the constraints is of most importance for the efficient use of gOcad. The primary types of constraints in gOcad include the following: *constraints on border*, *constraint points*, *constraint nodes*, *constraints on surface*, *vectorial link constraints*, *thickness constraints*, and *range thickness constraints*. The concepts of *direction of shooting*, *shooting point*, and *impact point* are important, as are the optimized shooting direction and the given shooting direction. The constraints are always established for all of the nodes of an object, but they can be sequentially modified (by changing the shooting direction or by deactivating them) one by one by the local constraint editing functions.

Normally, gOcad uses a “smooth” object to represent a geo-object which is modeled from raw data. Occasionally, an object can be directly manipulated by tools, such as *node/triangle/border extremity/border move/bridge/break/collapse/extend* (see (Paradigm-GOCAD, 2014)). The “part/fill holes” function is also useful. To build the 3D models, knowledge of the geological feature/classification is required.

### 2.4.5 Discrete smooth interpolation (DSI)

In discrete models, each object is modeled as a set of interconnected nodes, maintaining the geometry and the physical properties of the object. The topology of an object can be approximated by a graph  $G(\Omega, N)$  where  $\Omega$  is the set of all of the nodes of the graph, each of these nodes being identified with its index  $\Omega = \{1, 2, \dots, \alpha, \dots, M\}$ , and  $N$  is a map from  $\Omega$  into a subset of  $\Omega$  such that  $\beta \in N(\alpha)$  indicates that  $\beta$  can be reached in at most  $s(\alpha)$  steps from  $\alpha$  where  $s(\alpha) > 0$  is a given function of the node  $\alpha$ . The graph  $G(\Omega, N)$  is always assumed symmetrical, i.e.,  $\beta \in N(\alpha)$  if and only if  $\alpha \in N(\beta)$ . The topological model  $G(\Omega, N)$  is embedded into 3-dimensional Euclidean space by three functions  $\{\varphi^x(\alpha), \varphi^y(\alpha), \varphi^z(\alpha)\}$ , corresponding to the location of the node  $\alpha \in \Omega$ . To generalise these functions, each node  $\alpha \in \Omega$  is associated with a series of  $n$  functions,  $\varphi(\alpha) = \{\varphi^1(\alpha), \dots, \varphi^v(\alpha), \dots, \varphi^n(\alpha)\}$ , and some functions encode the coordinates to represent the geometry of the object, whereas other functions encode the physical properties of the object. To model natural objects, another component, called the constraint set  $C$ , must be added to the model with respect to the influence of the heterogeneous input data on the model. Finally, the discrete model, denoted as  $M^n(\Omega, N, \varphi, C)$ , consists of a triplet composed of

$G(\Omega, N)$ , a series of functions  $\varphi$ , and a set of constraints  $C$ .

The set of constraints  $C = \{c_1, c_2, \dots\}$  is divided into the three subsets  $C^\simeq, C^=, C^>$ . Each constraint  $c \in C$  is assumed to be linear and to have one of the following three general forms where  $\{A_c^\nu(\alpha)\}$  and  $b_c$  are given coefficients defining the constraint  $c$ :

$$\begin{aligned} \{c \in C^\simeq \text{honoured}\} &\Leftrightarrow \sum_{\alpha \in \Omega} \sum_{\nu} A_c^\nu(\alpha) \cdot \varphi^\nu(\alpha) \simeq b_c, \\ \{c \in C^= \text{honoured}\} &\Leftrightarrow \sum_{\alpha \in \Omega} \sum_{\nu} A_c^\nu(\alpha) \cdot \varphi^\nu(\alpha) = b_c, \\ \{c \in C^> \text{honoured}\} &\Leftrightarrow \sum_{\alpha \in \Omega} \sum_{\nu} A_c^\nu(\alpha) \cdot \varphi^\nu(\alpha) \geq b_c. \end{aligned} \quad (2.3)$$

These constraint subsets can be considered as follows. Subset  $C^\simeq$  is the set of “soft” equality constraints that must be honoured in a least square sense; subset  $C^=$  is the set of “hard” equality constraints that must be strictly honoured, and subset  $C^>$  is the set of “hard” inequality constraints that must be strictly honoured.

The Discrete Smooth Interpolation (DSI) has been specifically designed for interpolating the series of functions  $\varphi$  of a discrete model  $M^n(\Omega, N, \varphi, C)$  while considering all of the constraints  $c \in C$ . The DSI algorithms converge towards a solution of minimising a quadratic objective function  $R^*(\varphi)$ , the generalised roughness, which is defined as follows:

$$R^*(\varphi) = \sum_{\alpha \in \Omega} \mu(\alpha) \cdot R(\varphi|\alpha) + (\phi \cdot \omega) \cdot \sum_{c \in C^\simeq} \omega_c \cdot \rho(\varphi|c) \quad (2.4)$$

where,  $R(\varphi|\alpha)$  is the local roughness at node  $\alpha$ ,  $\rho(\varphi|c)$  is the local degree of violation of  $c$  by  $\varphi$  for each soft constraint  $c$ ,  $\mu$  is a stiffness coefficient, and  $\omega_c$ ,  $\phi \cdot \omega$  are weight coefficients ( $\omega$  is the balancing ratio and  $\phi$  is the fitting factor). DSI is largely inspired by cubic splines and inquiries from geoscience.

## 2.5 Surface representations

### 2.5.1 Mathematical surface representations

A 2-dimensional surface embedded in the 3-dimensional Euclidian space (3D space) can be given in one of three forms as follows:

- Explicit form:  $z = f(x, y)$ .
- Implicit form:  $F(x, y, z) = 0$ .
- Parametric form:  $x = x(u, v), y = y(u, v), z = z(u, v)$ .

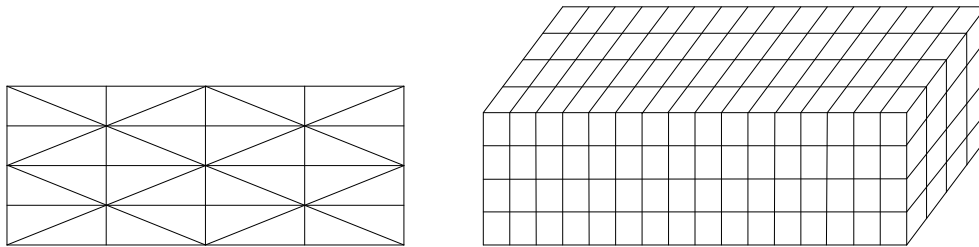
In technical applications, the explicit form of a surface is often difficult to determine. Moreover, the explicit form can be considered a special type of the parametric form in which  $x = u, y = v, z = f(u, v)$ . Therefore, a significant amount of literature only distinguishes two main classes of surface representations, i.e., parametric representations and implicit representations (Botsch et al., 2010). In the parametric representation, a surface is defined by a vector-valued parameterisation function  $f : \Omega \rightarrow S$  that maps a 2D parameter domain  $\Omega \in R^2$  to the surface  $S = f(\Omega) \in R^3$ . In implicit representation, a surface is defined to be the iso-level of a scalar-valued function  $F : R^3 \rightarrow R^2$ , i.e.,  $S = \{x \in R^3 | F(x) = a\}$ .

### 2.5.2 The surface approximation

Instead of having given surfaces in an analytical formula, most surfaces are given by sample points and then the applications build up surfaces by either interpolation or triangulation. Therefore, in general, only an approximation of the surface can be found in applications. Because polynomial functions are efficiently evaluated by arithmetic operators, polynomial functions are the natural choice for the approximation. Moreover, according to Taylor's theorem, the asymptotic approximation error is  $O(h^{p+1})$ , where  $h$  is the diameter of the domain and  $p$  is the degree of the polynomial (Botsch et al., 2010).

Generally, there are two methods used to improve the accuracy of the approximation, as follows: (1) raising the degree of the polynomial (*p-refinement*) and (2) reducing the size of the domain (*h-refinement*), i.e., by splitting the domain into smaller segments (Figure 2.13). In h-refinement, for each segment, a poly-

nomial (a patch) is defined to locally approximate the part of the surface in the segment and to be sufficiently smooth with the neighbour patches. Because processing a large number of simple objects is often faster than processing a smaller number of more complex objects, many geometry processing applications prefer h-refinement to p-refinement, although they accept  $C^0$  piecewise linear surface representations, i.e., polygonal or triangular meshes, as the ad-hoc standard. For implicit representations, a constant-value in each segment (cell) is also accepted as the ad-hoc standard in many technical applications. The remaining issue is the sufficient management of the segments (patches), i.e., small storage with a fast access and rapid processing. Studies on data structures have been dedicated to this issue.



**Figure 2.13:** Splitting the domain into smaller segments

### 2.5.3 Conversions between representations

#### Explicit representations to implicit representations

The conversion from an explicit to an implicit representation includes computation of a signed distance field. When implicit representations use the 3D regular grid structure, the efficient algorithms for the conversion include those presented by Kaufman (1987). When parametric representations are presented as triangulated meshes, the distance of a grid node to a given mesh is the distance to the closest triangle. Computing this distance can be efficiently performed using spatial data structures, such as octree, k-d tree, AABB tree, locality-sensitive hashing and many modifications of these structures (Bentley, 1975, CGAL, 2014, Datar et al., 2004, Liaw et al., 2010, Samet, 1994, Zatloukal et al., 2002). The sign of the distance field, which determines whether a grid node lies inside or outside the object, is defined by the normal vector of the closest triangle of the node. Sethian (1996) proposed a method called fast marching to accelerate the

computations of the distances in the entire grid.

### **Implicit representations to explicit representations**

The conversion from an implicit to an explicit representation is known as an iso-surface extraction and is primarily used to convert volumetric representations or 3D images to triangulated meshes. The volumetric representation, which represents the scalar functions  $f : R^3 \rightarrow R$  is often used in many modelling, analysis and simulation problems in the geosciences. There have been many studies on iso-surface extraction from this structure for the purpose of visualisation to provide insight into the data. Iso-surface extractions can also be used to construct data for spatio-temporal information systems.

Studies on iso-surface extractions can generally be divided into two approaches. The first approach includes the marching cubes algorithm and its variants (Bischoff et al., 2005, Kobbelt et al., 2001, Lorensen and Cline, 1987). These algorithms examine each edge of the cell that intersects the iso-surface  $S$ . When an edge with two endpoints  $p_1, p_2$  and the values of the scalar  $f$  at these points  $d_1 = f(p_1)$ ,  $d_2 = f(p_2)$  differ in sign, i.e.,  $d_1 * d_2 < 0$ , then the intersection point,  $s$ , can be defined by a linear interpolation, according to the following equation:

$$s = \frac{|d_2|}{|d_1| + |d_2|}p_1 + \frac{|d_1|}{|d_1| + |d_2|}p_2. \quad (2.5)$$

The intersection points of each cell are then connected to a triangulated surface patch based on 256 standard configurations (that can be deduced from 15 base configurations). The collection of all of the patches yields a triangulated mesh approximation of the iso-surface  $S$ .

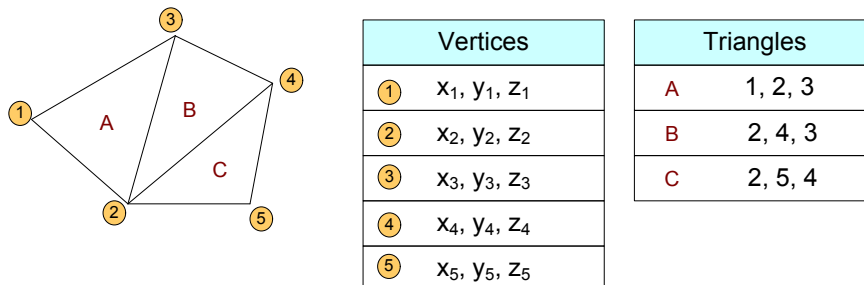
In the second approach, algorithms based on a provable sampling technique are used, and the samples are then triangulated to obtain the iso-surfaces (Boissonnat and Oudot, 2005, CGAL, 2014). This approach prevent the problem of high triangle complexity in marching cubes-like approaches. The resulting surface contains only well-shaped triangles and faithfully approximates the input surface.

### 2.5.4 Triangulated surface data structures

Data structures organise data in certain configurations so that these data can be captured, stored, and manipulated effectively and efficiently. Unfortunately, a data structure does not always simultaneously satisfy these requirements. Some data structures are simple to construct and have a small storage but are not sufficiently fast to access the data. Other data structures explicitly store the topology, i.e., the incident relationships between the vertices, edges, and faces. Using these data structures, the processing algorithms can have direct access to the neighbourhood of a vertex, an edge, or a face. However, these data structures are difficult to construct and use large storage. Several typical data structures are briefly reviewed below.

#### Indexed-face-set data structure

The simplest way to represent a triangulated surface is to store its vertices and its faces (by referencing to the vertices), called an indexed-face-set data structure. An example of this structure is shown in Figure 2.14.



**Figure 2.14:** An example of indexed-face-set data structure

Recall the famous Euler formula about the relationship between the number of vertices  $V$ , edges  $E$ , and faces  $F$  of a closed and connected mesh is  $V - E + F = 2(g - 1)$ , where  $g$  is the genus of the surface. Because each triangle is bound by three edges, each interior edge (of the manifold surface) is incident to two triangles, and  $g$  is often a small number, the following formulas can be derived for triangulated mesh:

$$F \approx 2V; E \approx 3V; \text{ the average number of incident edges of each vertex is } 6.$$

If each vertex coordinate uses a 32-bit single precision number, then each vertex is stored in 12 bytes. If each vertex index uses a 32-bit integer number, then each

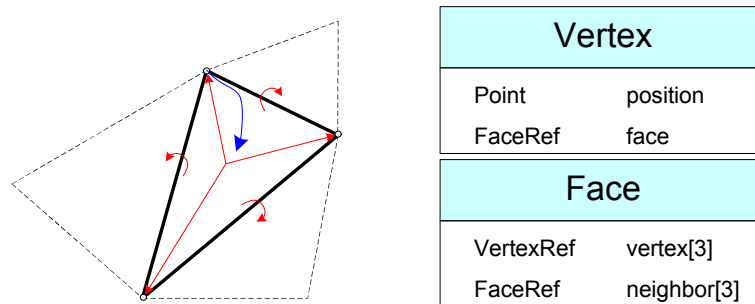


triangle uses 12 bytes. Generally, storing a surface requires 36 times the number of vertices of bytes memory.

Because of the simplicity and efficiency in the storage of the indexed-face-set data structure, it is used in file formats, such as OFF, OBJ, and VRML. The indexed-face-set data structure is also a favourite for representing triangulated surfaces in database systems (GiGa-Infosystems, 2014).

### Face-based with connectivity information

The face-based with connectivity information data structure is used for the 2D triangulation data structures of CGAL (CGAL, 2014). In this data structure, each triangle stores references to its three vertices and its three neighbouring triangles; each vertex stores a reference to one of its incident triangles (Figure 2.15).



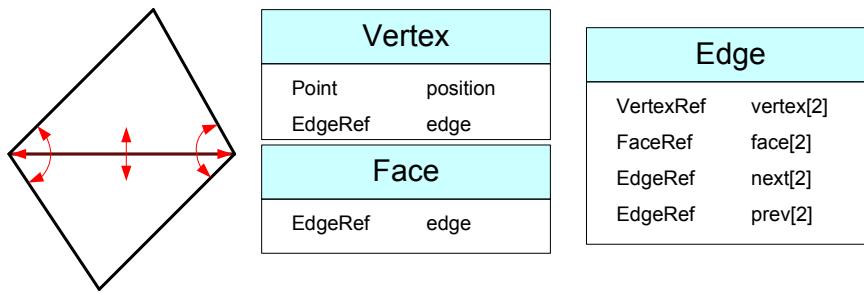
**Figure 2.15:** Face-based with connectivity information data structure. Each vertex stores its position and a reference to one of its incident faces. Each face stored references to its three vertices and its three neighboring triangles

### Winged-edge data structure

The winged-edge data structure was invented by Baumgart (1972). In this data structure, each edge stores references to its two vertices, its two incident faces, its two next edges on the left and right faces, and its two previous edges on the left and right faces; each vertex and each face refers to one of its incident edges (Figure 2.16).

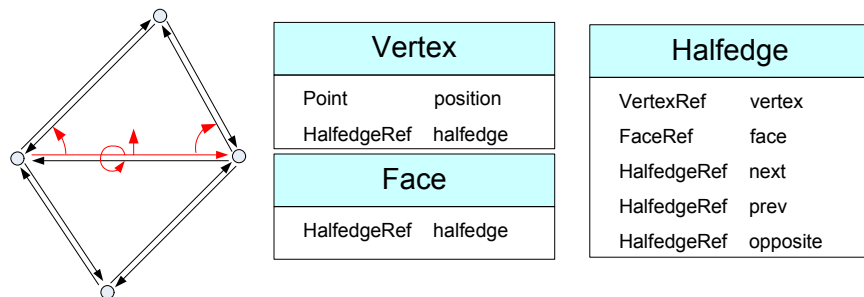
### Halfedge data structure

The halfedge data structure has been presented in the literature (Kettner, 1999, Weiler, 1985), and has been used as the package of the same name in CGAL



**Figure 2.16:** Winged-edge data structure. Each vertex and each face refers to one of its incident edges. Each edge stores references to its two vertices, its two incident faces, its two next edges on the left and right faces, and its two previous edges on the left and right faces

(CGAL, 2014). The data structure is shown in Figure 2.17. In this data structure, the *halfedge* is an edge that is consistently oriented in the counterclockwise order around each face and along each boundary. Each halfedge refers to the vertex it points to, its adjacent face (a zero pointer if it is a boundary halfedge), the next halfedge of the face or boundary (in the counterclockwise direction), the previous halfedge in the face, and its opposite (or inverse) halfedge.



**Figure 2.17:** Halfedge data structure. Each vertex stores an outgoing halfedge. Each face stores a reference to one of its halfedges. Each halfedge stores references to the vertex it points to, its adjacent face, the next halfedge of the face or boundary, the previous halfedge in the face, and its opposite halfedge.

Among surface data structures mentioned above, halfedge data structure is the most flexible structure. It represents each of the mesh items (vertices, edges, and faces) explicitly, in order to be able to attach additional attributes and functionality to them. Moreover, it is easy to travel between vertices, edges, and faces.

# Chapter 3:

## Data Models

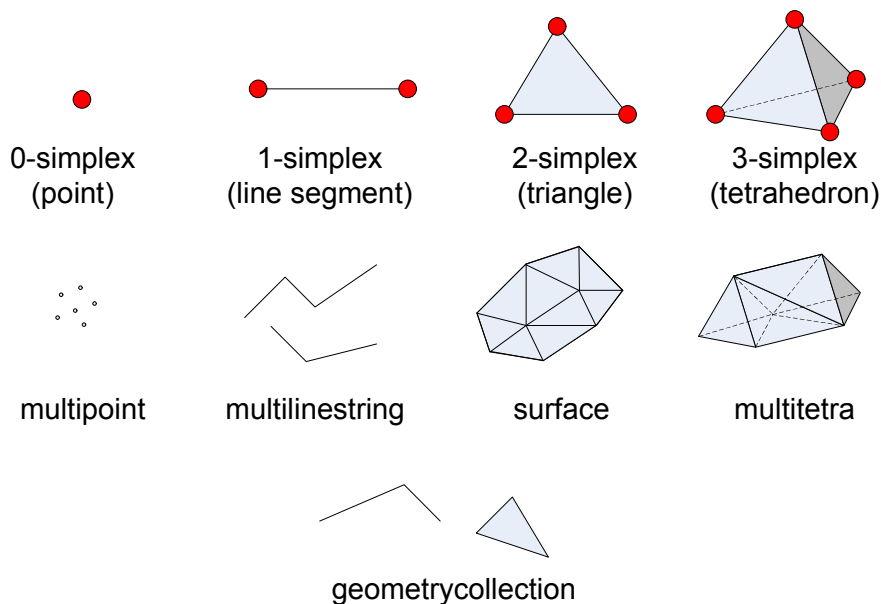
---

In spatio-temporal geoscience information systems (TGSIS), the data are stored in database management systems (DBMS). Using DBMS, TGSIS gain many of the advantages of the DBMS, such as multi-users, concurrent access control, security, transactions, among others. In this context, data models are object-relational database schemata that define the structures and integrity constraints imposed on the databases. Five data models are presented in this chapter. The first data model represents geo-objects with the assumption that they are static, i.e., they do not change over time. The geo-objects include their geometries in 3-dimensional space and their physical properties in the components. In the second data model, geological surfaces are represented as constants in each period. The third data model represents geological surfaces by piecewise linear functions of time. In the framework of this thesis, we have published an object-relational spatio-temporal geoscience data model, the so-called TGSIS data model. TGSIS data model is a general data model for geo-objects with an  $n$ -dimensional geometry embedded in the  $m$ -dimensional Euclidian space  $R^m, m \geq n$ . This model represents geo-objects that evolve continuously in one-dimensional valid time. The fifth data model represents multiple indexed geoscience data in which time and other non-spatial dimensions are interpreted as larger spatial dimensions.

This chapter is organised as follows. Sections 3.1, 3.2, 3.3 present the first, second, and third data models, respectively. Section 3.4 reviews the TGSIS data model. The fifth data model is presented in Section 3.5. Section 3.6 is dedicated to the summary.

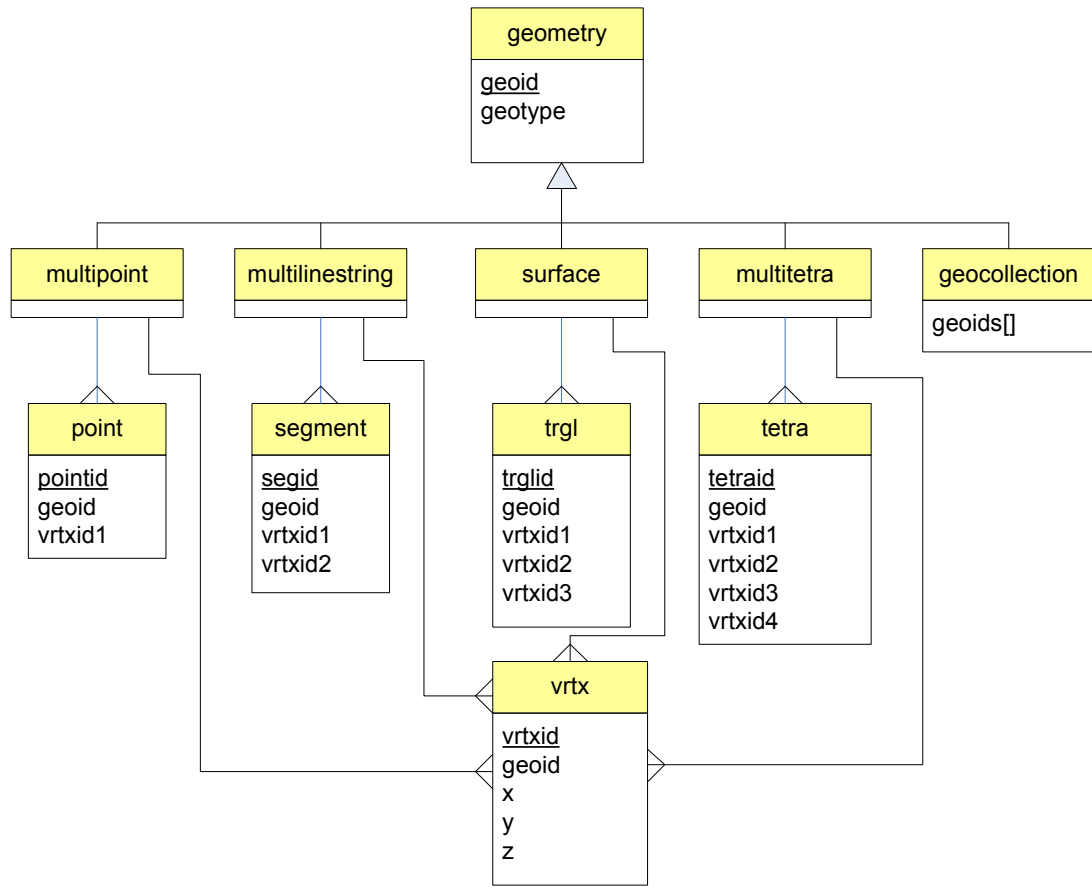
### 3.1 A 3D spatial data model

Geological objects can simply be abstracted by geometric objects in the 3-dimensional space. Theoretically, geometric objects can be partitioned and approximated by a set of simplices equipped with appropriate relationships. The simplices in the 3-dimensional space are as follows: 0-simplex (point), 1-simplex (line segment), 2-simplex (triangle), and 3-simplex (tetrahedron), and the sets of these simplices are denoted as *multipoint*, *multilinestring*, *surface*, and *multitetra*, respectively. A point can be represented by its three coordinates,  $x$ ,  $y$ , and  $z$ , called a vertex. A line segment, a triangle, and a tetrahedron can be represented by two, three, or four vertices, respectively. The vertex object class (VRTX) is used to store positions, and then each point, line segment, triangle, and tetrahedron are represented, respectively, by one, two, three, or four pointers to the positions. Some appropriate operations, such as a buffer or intersection, can be applied to the geometric objects. To let the set of geometric objects closed under these operations, the geometry collection object class (GEOCOLLECTION) is added into data model. A geometry collection object is a set of other geometry objects. Figure 3.1 shows the simplices and the geometry objects.



**Figure 3.1:** Simplices and geometry objects

An object-relational data schema of the geometry objects is shown in Figure 3.2.

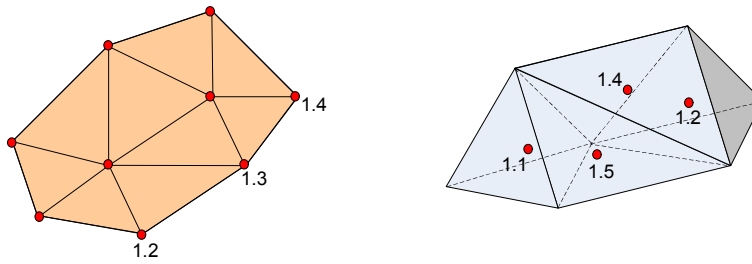


**Figure 3.2:** Object-relational data schema of the geometry objects.

In the above data schema, the *geotype* attribute can receive the following values: “GEOMETRY”, “MULTIPOINT”, “MULTILINESTRING”, “SURFACE”, “MULTITETRA”, and “GEOMETRYCOLLECTION”. These values are used to determine the type of concrete geometry object. The topology of an object is not explicitly stored, but this information can be deduced from the geometry information when software systems load the object into the memory.

In many geoscience applications, physical properties, such as porosity and permeability, are required to attach to vertices of a surface or to tetrahedrons of a *multitetra* (cells of a solid), as shown in Figure 3.3.

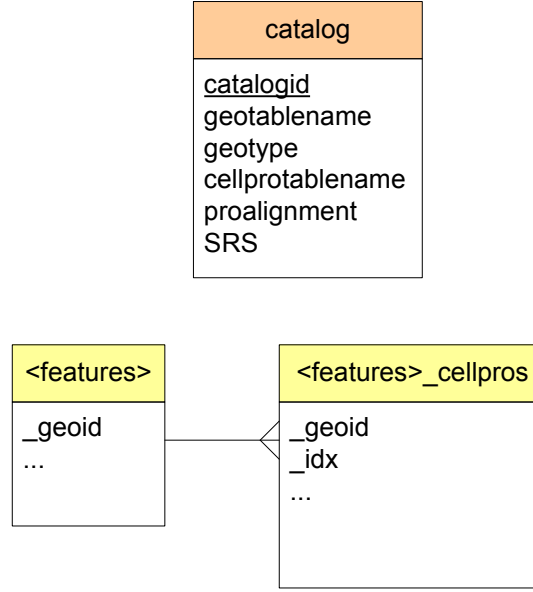
We use the term *feature* to denote a geological object that has geometry, topology, and properties. The term feature class denotes the set of features with the same type of geometry, i.e., *multipoint*, *multilinestring*, *surface*, *multitetra*, *geometrycollection*, the same set of properties defined by names and types, and the same type of attachment of the properties to the geometry, i.e., attaching to the



**Figure 3.3:** An example of attaching the porosity properties onto vertices and cells

vertices or the cells. A feature class is represented by two tables and defined by applications called  $\langle features \rangle$  and  $\langle features \rangle\_CELLPROS$ . In Figure 3.4, the name  $\langle features \rangle$  can be substituted by any appropriate table that is name defined by the applications. The only requirement of these tables is the  $\_geoid$  attribute in the table  $\langle features \rangle$  and the  $\_geoid$  attribute and the  $\_idx$  attribute in the table  $\langle features \rangle\_CELLPROS$ . This requirement is automatically satisfied by the software described in Chapter 6. The  $\_idx$  attribute can receive the value of the attributes  $VRTXID$ ,  $POINTID$ ,  $SEGID$ ,  $TRGLID$ , and  $TETRAID$ , depending on the type of geometry and the manner in which the physical properties attach to the geometry defined by the *feature class*. Other attributes in the table  $\langle features \rangle$  represent the physical properties of each feature in the *feature class*. Other attributes in table  $\langle features \rangle\_CELLPROS$  represent the properties attaching to each vertex or cell of a feature in the feature class. The goal of the table CATALOG is to store information defined for each *feature class*; this information is the name of the  $\langle features \rangle$  (*geotablename*), the type of geometry (*geotype*), the name of the  $\langle features \rangle\_CELLPROS$  (*cellprotablename*), the manner in which the physical properties attach to the geometry (*proalignment*), and the identifier of the spatial reference system (*SRS*). The *proalignment* attribute can receive one of the following two values: “VERTEX” or “CELL”. Figure 3.4 shows the data schema for the properties required to add to the geometry data schema mentioned above.

For example, an application must manage the geological horizon surfaces with the name of the interpreter, certain metadata, and the physical properties of porosity and permeability measured at each vertex of the surface. The data schema can be defined as follows: Table *HORIZONS*( $\_geoid$ , *creator*, *note*); table *HORIZONS\_cellpros*( $\_geoid$ ,  $\_idx$ , *porosity*, *permeability*); the values in the table CATALOG include “HORIZONS” for *geotablename*, “SURFACE” for *geotype*, “HORIZONS\_cellpros” for *cellprotablename*, and “VERTEX” for *proalignment*.



**Figure 3.4:** Data schema for the properties

## 3.2 A data model for multi-instance surfaces

The simplest way to expand a data model into the time domain is to add a time property to the model. Therefore, the model represents objects at points of time (instances). Assume that a geological surface is defined at time instances,  $t_0$ ,  $t_1$ , and  $t_2$ ,  $t_0 < t_1 < t_2$ , denoted as  $G_0$ ,  $G_1$ , and  $G_2$ , respectively. We also denote  $G(t)$  as a representation of the surface in the interval  $[t_0, t_2]$ . Certain approximations of  $G(t)$  can be defined by Equation 3.1 or Equation 3.2), as follows:

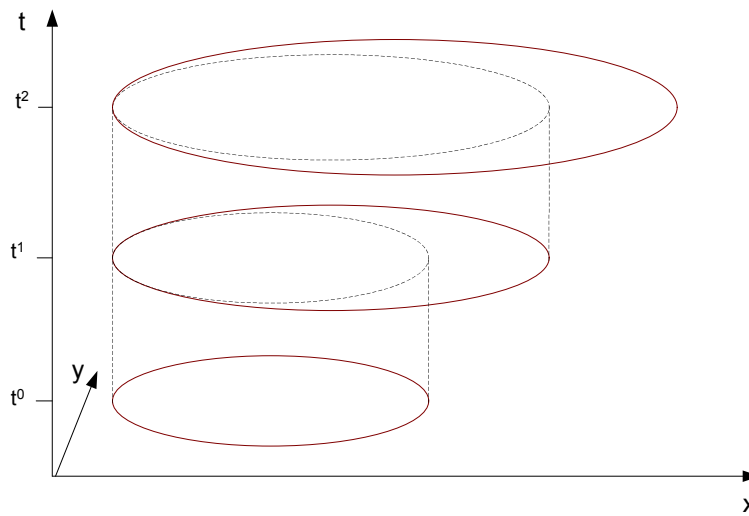
$$G_1(t) = \begin{cases} G_0 & \text{if } t_0 \leq t < t_1 \\ G_1 & \text{if } t_1 \leq t < t_2 \\ G_2 & \text{if } t = t_2 \end{cases} \quad (3.1)$$

or

$$G_2(t) = \begin{cases} G_0 & \text{if } t = t_0 \\ G_1 & \text{if } t_0 < t \leq t_1 \\ G_2 & \text{if } t_1 \leq t < t_2 \end{cases} \quad (3.2)$$

Figure 3.5 shows a graphical representation of Equation 3.1.

The table TIMES is added to represent the sequence of time instances for each object. This table contains the *times* attribute equipped with the *geoid* attribute to uniquely define a geometry of the object at a specific time. A geological



**Figure 3.5:** A graphic representation of Equation 3.1

object can have certain properties, such as fault throw and thickness, that have changing values at each time instance. Table  $\langle features \rangle\_PROS$  defined by the application is used to store these types of properties. The *protablename* attribute in table CATALOG is also added to maintain the information of the name of the  $\langle features \rangle\_PROS$  table defined by a feature class. Finally, the data schema for geological surfaces in multi-time instances is shown in Figure 3.6.

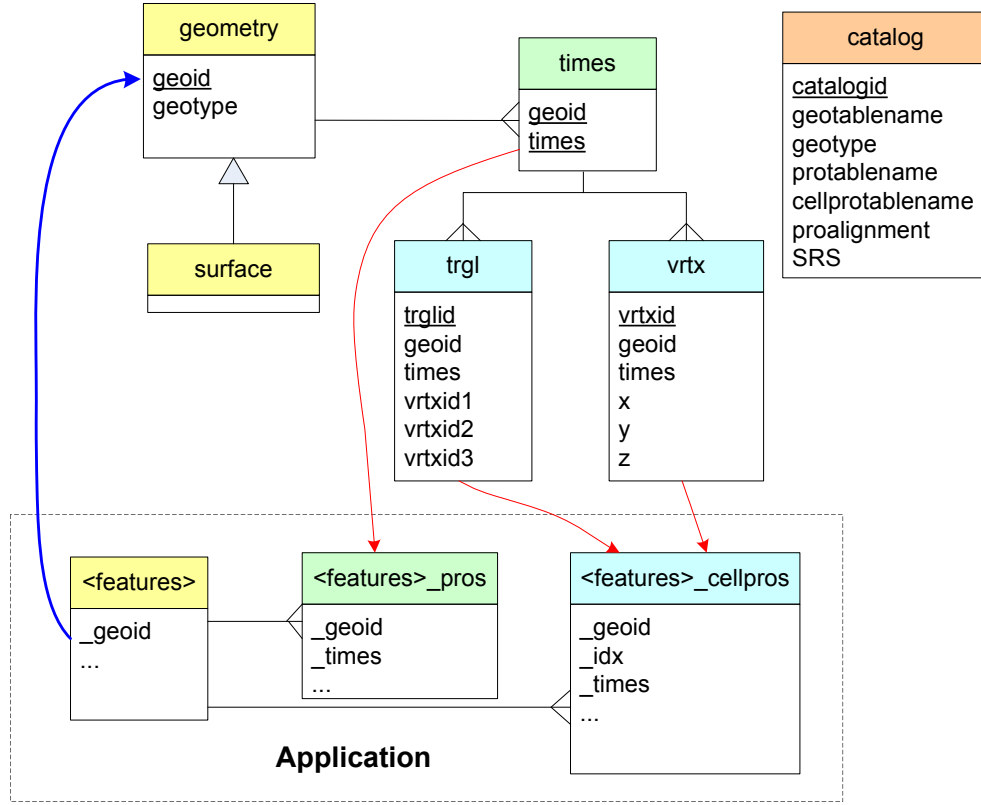
### 3.3 A data model for continuously evolving surfaces

There are many geological processes, such as deformation, deposition, erosion in which objects are continuously change. The data model for multi-instance surfaces mentioned above is unsuitable to represent these processes. The new model must represent objects in the time interval  $[t_0, t_1]$  using a continuous function. The simplest function is the linear function, i.e., the function

$$f(t) = f(t_0) * \frac{t_1 - t}{t_1 - t_0} + f(t_1) * \frac{t - t_0}{t_1 - t_0} \quad (3.3)$$

for all values  $t \in [t_0, t_1]$ . Moreover, while evolving in the time interval  $[t_0, t_1]$ , several geometrical or physical constraints are always imposed on the objects. To model the objects in these processes more accurately, the selected continuous function is a piecewise linear function with pre-defined values at  $g_0 \geq 1$  time





**Figure 3.6:** Data schema for geological surfaces in multi-time instances

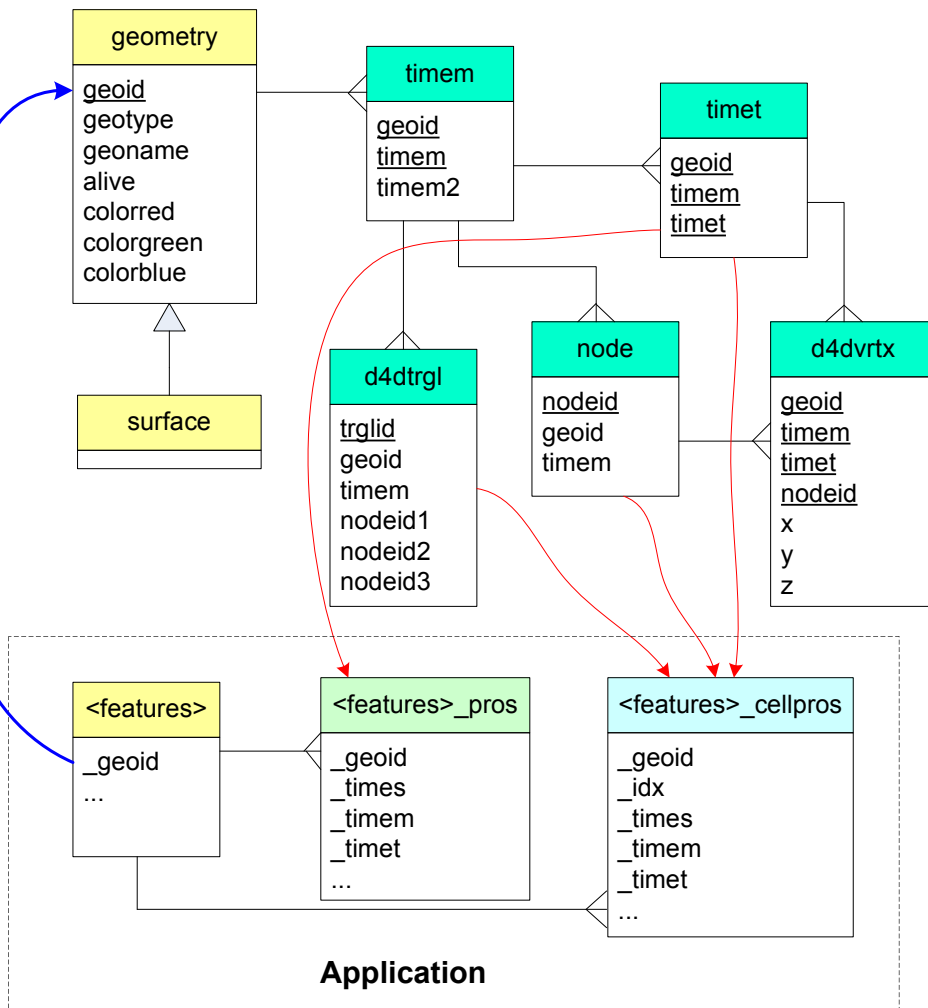
points,  $\{t_0^0, t_0^1, \dots, t_0^{g_0}\} \in [t_0, t_1]$  where  $t_0^0 = t_0$  and  $t_0^{g_0} = t_1$ . The algorithms used to determine this function, based on two instances of the object at time  $t_0$  and  $t_1$ , are called morphological interpolations, as shown in Chapter 4.

Assume that an object is defined at a sequence of time instances denoted as  $T^M = \{t_0, t_1, \dots, t_k\}$  for a given integer number  $k > 0$ . Using morphological interpolations, the object is defined in the collection of time instance sequences

$$T^G = \{\{t_0^0, t_0^1, \dots, t_0^{g_0}\}, \{t_1^0, \dots, t_1^{g_1}\}, \dots, \{t_{k-1}^0, \dots, t_{k-1}^{g_{k-1}}\}, \{t_k^0\}\}$$

where  $t_0^0 = t_0$ ,  $t_i^{g_i} = t_{i+1}^0 = t_{i+1}$  for all  $i = 1, \dots, k-1$ . In each sequence  $\{t_i^0, \dots, t_i^{g_i}\}$ , the topology of the object is unchanged; only the geometry of the object changes. In the case in which the geometry of the objects is represented by a triangulated mesh, each triangle is completely determined by its three vertices, the structure of the triangle remains unchanged, and only the coordinates of the vertices change. Therefore, we model surfaces in the time domain using the following three tables:  $\text{NODE}(\text{nodeid}, \text{geoid}, \text{timem})$ ;  $\text{D4DTRGL}(\text{trglid}, \text{geoid}, \text{timem}, \text{nodeid1}, \text{nodeid2}, \text{nodeid3})$ ; and  $\text{D4DVRTX}(\text{geoid}, \text{timem}, \text{timet},$

$nodeid, x, y, z$ ). Table TIMEM represents the sequence  $T^M$ , and table TIMET represents the collection of sequences  $T^G$ . The attributes  $\_timem$  and  $\_timet$  are also added to the tables  $\langle features \rangle\_PROS$  and  $\langle features \rangle\_CELLPROS$  to maintain the relationships between the geometry and the properties of the geological surfaces. Because we typically run queries about an object at time points after all of its defined time instances, i.e., after  $t_k^{gk}$ , we assume that if the queried object still exist at the moment, then its geometry, topology, and other properties are as defined at time  $t_k^{gk}$ . The *alive* attribute in the table GEOMETRY aims to determine whether an object exists. To support the display of the object in software systems, the attributes *geoname*, *colorred*, *colorgreen*, and *colorblue* are added to the table GEOMETRY. Figure 3.7 shows the data schema for the geological surfaces continuously evolving in the time domain.



**Figure 3.7:** Data schema for geological surfaces continuously evolving in the time domain

## 3.4 The TGSIS data model

Within the scope of this study, we have developed an object-relational spatio-temporal geoscience data model, called TGSIS data model (Le et al., 2013). The model is developed for spatially and temporally indexed multi-dimensional geoscience data by first embedding a combinatorial topological model in terms of G-Maps in the domain  $R^m \times Time$  ( $m \in N$ ), and then converting it into an object-relational model which can easily be implemented in an object-relational database system. In this section, we will review the underlying concepts of this data model. Note that we substitute the term GST by the term TGSIS to be consistent with other chapters in this dissertation.

### 3.4.1 Geometric modeling based on topology – generalised maps

#### Modeled geometric objects

The model represents the basic geometric objects which are subsets of  $m$ -dimensional Euclidean space described by  $n$ -dimensional quasi-manifolds ( $0 \leq n \leq m$ ). Before giving a definition of quasi-manifold, some terms need to be recalled.

Let  $i$  be an integer,  $i > 0$ ,  $R^i$  be the  $i$ -dimensional Euclidean space, and  $||\cdot||$  be the standard norm in  $R^i$ , it is called  $B^i = \{x \in R^i : ||x|| < 1\}$  as *open  $i$ -ball*,  $\overline{B^i} = \{x \in R^i : ||x|| \leq 1\}$  as *closed  $i$ -ball*, and  $S^i = \{x \in R^{i+1} : ||x|| = 1\}$  as *standard  $i$ -sphere*. For convenience, the *open 0-ball*  $B^0$  (*closed 0-ball*  $\overline{B^0}$ ) is considered as a single point.

For  $i \geq 0$ , an *open* (a *closed*)  *$i$ -cell* is a set homeomorphic to an open (a closed)  *$i$ -ball*; an  *$i$ -sphere* is a set homeomorphic to a standard  *$i$ -sphere*. The dimension of an open (a closed)  *$i$ -cell* (or an  *$i$ -sphere*)  $c$  is  $i$ , denoted as  $dim(c)$ .

**Definition 3.1** *Let  $n$  be an integer,  $n \geq 0$ , an  $n$ -dimensional finite CW(closure-finite, weak topology)-Complex is a pair  $(X, \mathcal{P})$  where  $X$  is a Hausdorff space and  $\mathcal{P}$  is a finite partition of  $X$  into open cells of dimension not greater than  $n$ , and for every open  $i$ -cell  $c$ ,  $0 < i \leq n$ , in  $\mathcal{P}$ , exists a continuous map  $H$  from the closed  $i$ -ball to  $X$  such that*

- (i) *the restriction to the open  $i$ -ball is a homeomorphism onto  $c$ , and*

- (ii) the image of the restriction to the standard  $(i - 1)$ -sphere is the union of open cells in  $\mathcal{P}$  of dimension less than  $i$ .

Lemma 1 in Le et al. (2013) shows that if  $i > 0$ , the image of the restriction of  $H$  to the standard  $(i - 1)$ -sphere is independent of  $H$ , and therefore, we can define the *boundary* of an  $i$ -cell  $c$ , denoted  $\partial c$ , as this image,  $H(S^{i-1})$ . For convenience, the boundary of 0-cell is empty.

Using the term boundary, terms *face*, *co-face*, *incidence*, *adjacency*, *link*, *connectivity* can be defined. Given a cell  $\alpha$ , a cell  $\beta$  is called a face of  $\alpha$  if and only if  $\beta \in \partial\alpha$ ; in this case,  $\alpha$  is called co-face of  $\beta$ . Two cells  $\alpha, \beta$  are incident if and only if either  $\alpha$  is a face of  $\beta$  or  $\beta$  is a face of  $\alpha$ . Cells  $\alpha, \beta$  are adjacent if and only if  $\dim(\alpha) = \dim(\beta)$  and there exists a cell  $\gamma$ , which is a face of both  $\alpha$  and  $\beta$ . Given a cell  $\alpha$ , a link of  $\alpha$  is a union of all boundaries of co-faces of  $\alpha$ , which is not incident to  $\alpha$ . An  $n$ -dimensional finite CW-Complex is connected if for every two 0-cells  $\alpha, \beta$  a sequence  $\gamma_0, \gamma_1, \dots, \gamma_k$  exists such that  $\alpha = \gamma_0, \beta = \gamma_k$  and  $\gamma_i, \gamma_{i+1}$  are incident for any  $0 \leq i < k$ .

**Definition 3.2** Let  $n$  be an integer,  $n \geq 0$ , an  $n$ -dimensional quasi-manifold is an  $n$ -dimensional finite CW-Complex  $(X, \mathcal{P})$  which satisfies the following conditions:

- (i) *Finite, regular CW-Complex:* Every cell is either an  $n$ -cell or a face of an  $n$ -cell.
- (ii) *Pseudo-manifold:* If  $n > 0$ , every  $(n - 1)$ -cell is a face of at most two  $n$ -cells.
- (iii) *Quasi-manifold:* Inductively, 0 and 1-pseudo-manifolds are quasi-manifolds, for  $i > 1$ , a link of each 0-cell is a connected  $(i - 1)$ -quasi-manifold.

In topological sense, 0-cells, 1-cells, 2-cells, 3-cells are called vertices, edges, facets, volumes respectively.  $i$ -cells describing cells in  $R^m$ ,  $0 \leq i \leq m$ , are called real  $i$ -cells, for example, 0-cells (points), 1-cells (lines), 2-cells (planar polygons), 3-cells (polytopes).

Symbol  $\mathbf{K}_i$  is used to define a set of  $i$ -cells contained in the partition  $\mathcal{P}$  of  $X$ .

$$\mathbf{K}_i = \{c \in \mathcal{P} : \dim(c) = i\}$$

We recall that a quasi-manifold is a decidable space of arbitrary dimension, and that the algorithm in Floriani et al. (2002) enables us to decompose any non-manifold into an assembly of quasi-manifolds.

### Generalised Maps (n-GMap)

Graph theory can be used to construct a graph, where vertices represent cells and edges represent incidence relationships between cells. Such a graph is called an incidence graph. Brisson (1989, 1993) introduced “Cell-Tuple-Structure”, where the incidence relationships are described in terms of cell-views  $v = \{c_i, c_{i+1}, \dots, c_j\}$ ,  $j > i$ , or the view of cell  $c_i$  from cell  $c_j$ . The cell-view  $v = \{c_0, c_1, \dots, c_n\}$  is a view of vertex  $c_0$  from a cell with the largest dimension  $c_n$  and called “vertex-view”.

The pair  $(w, w')$  of two vertex-views  $w$  and  $w'$ ,

$$\begin{aligned} w &= \{c_0, \dots, c_{i-1}, c_i, c_{i+1}, \dots, c_n\}, \\ w' &= \{c_0, \dots, c_{i-1}, c'_i, c_{i+1}, \dots, c_n\}, \end{aligned}$$

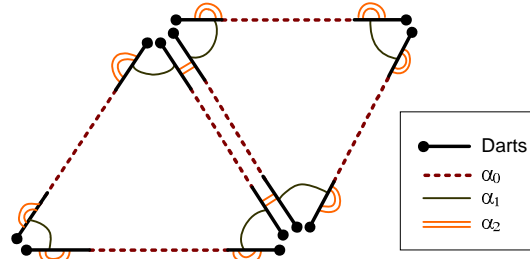
which share the same  $j$ -cells for all  $j \neq i$ , is called  $i$ -adjacent. Then  $i$ -adjacency induces a set of involutions  $\{a_i\}$  in the set of vertex-views.

Generalised maps (GMap) are similar to Brisson’s “Cell-Tuple-Structure”, but have been independently introduced by Lienhardt (cf. Lévy and Mallet, 1999, Lienhardt, 1989, 1994, Lienhardt et al., 2009, Mallet, 2002). GMap uses two objects “dart” and “ $i$ -involution”, where a dart denotes a path from an  $n$ -cell node to a 0-cell node of the incidence graph, and an  $i$ -involution is a mapping to generate an  $i$ -adjacency. Darts are analogous to vertex-views in Cell-Tuple structure. An involution is a bijection which is its own inverse. GMap is formally defined as follows (cf. Mallet, 2002).

**Definition 3.3** *An  $n$ -GMap of dimension  $n, n \geq 0$ , is an algebra  $\mathcal{M}(D, \alpha_0, \alpha_1, \dots, \alpha_n)$ , where  $D$  is a finite set of abstract elements called darts,  $\alpha_i, i = 0, \dots, n$ , are  $i$ -involutions on  $D$ , satisfying following conditions:*

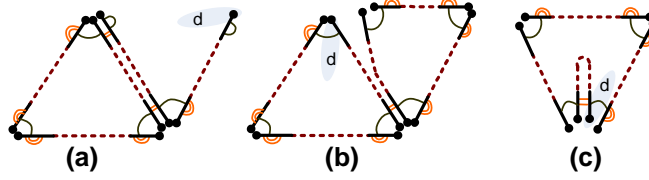
- (i)  $\alpha_i(d) \neq d \quad \forall d \in D; 0 \leq i < n;$
- (ii)  $\alpha_i \circ \alpha_{i+2+k} \{ \alpha_i \circ \alpha_{i+2+k}(d) \} = d \quad \forall d \in D; i \geq 0; k \geq 0; i + 2 + k \leq n;$
- (iii)  $\alpha_i \circ \alpha_{i+2+k}(d) \neq d \quad \forall d \in D; i \geq 0; k \geq 0; i + 2 + k \leq n.$

A graphical display of a 2-GMap is shown in Figure 3.8.



**Figure 3.8:** Graphical display of a 2-GMap

The three conditions of the definition of  $n$ -GMap ensure the consistency of boundary relationships and completeness of “darts sewing” when embedding geometric objects from  $n$ -GMap. Figure 3.9 shows examples of violations of these conditions.



**Figure 3.9:** Violations of the 1<sup>st</sup> condition (a), the 2<sup>nd</sup> condition (b), the 3<sup>rd</sup> condition (c)

An important operation on  $n$ -GMap is the “orbit”-operation defined as follows.

**Definition 3.4** An orbit  $\langle \alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_{i_k} \rangle (d)$  of a dart  $d \in D$  on a subset of involutions  $S = \{\alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_{i_k}\}$ , defined on the  $n$ -GMap  $\mathcal{M}(D, \alpha_0, \alpha_1, \dots, \alpha_n)$  is the set of all darts which are the image of  $d$  applying any combination of the involutions in  $S$ . Especially, an  $i$ -orbit of  $d$  is the orbit of  $d$  on all but the  $i$ -involution,  $\langle \alpha_0, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n \rangle (d) = \langle \phi_i \rangle (d)$ .

The  $i$ -orbit  $\langle \phi_i \rangle (d)$  is also referred as an “abstract”  $i$ -cell. In the next subsection we will show that a correspondence exists between an “abstract”  $i$ -cell and a “real”  $i$ -cell. Figure 3.10 shows some orbits of the example 2-GMap in Figure 3.8.

### Geometric Model (GModel)

$n$ -GMap and  $i$ -orbit operations determine a collection of sets of “abstract”  $i$ -cells  $\{\tilde{\mathbf{K}}_i\}$ , where  $\tilde{\mathbf{K}}_i = \{\tilde{c}_i = \langle \phi_i \rangle (d)\} \forall i \in [0, n]$ . The boundary operation of an

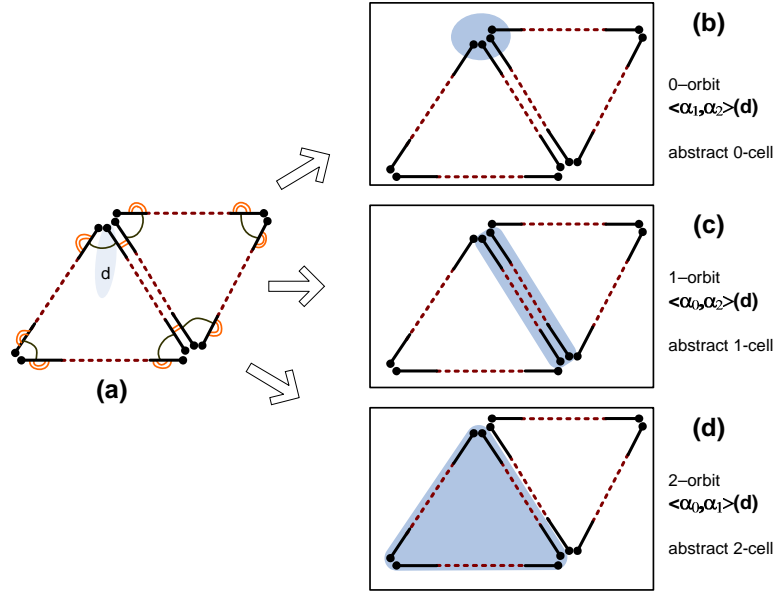


Figure 3.10: Orbits on the example 2-GMap

abstract  $i$ -cell  $\tilde{c}_i$  is induced from the definition of  $n$ -GMap as

$$\begin{aligned} \partial \tilde{c}_i &= \partial(\langle \phi_i \rangle (d)), \quad \text{for any } d \in \tilde{c}_i, \\ &= \{ \langle \phi_k \rangle (d') \mid 0 \leq k < i, d' \in \langle \phi_i \rangle (d) \}. \end{aligned}$$

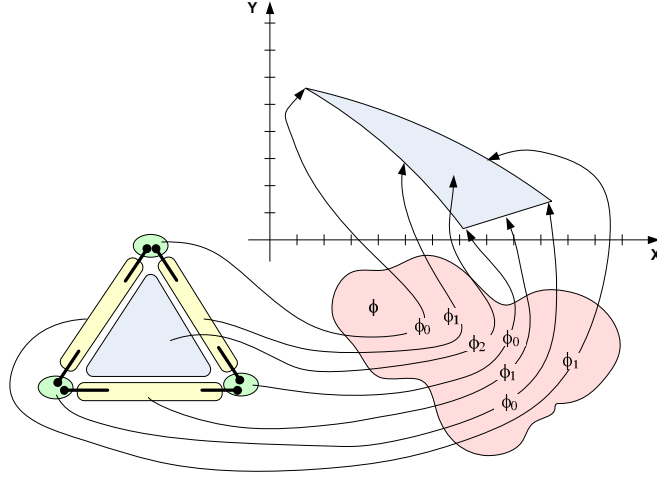
The boundary operation in  $\{\tilde{\mathbf{K}}_i\}$  enables us to define the incidence and adjacency relationships of two abstract cells analogously to the incidence and adjacency relationships of geometric cells.

- $\tilde{c}_i$  and  $\tilde{c}_j$  are incident if and only if either  $\tilde{c}_i \in \partial \tilde{c}_j$  or  $\tilde{c}_j \in \partial \tilde{c}_i$ .
- $\tilde{c}_i$  and  $\tilde{c}_j$  are adjacent if and only if  $i = j$  and there exists  $\tilde{c}_k$ ,  $k = i - 1$ , incident to both  $\tilde{c}_i$  and  $\tilde{c}_j$ .

To realize the abstract model in terms of GMap as a geometric object, each abstract  $i$ -cell is mapped onto a real  $i$ -cell in  $\mathbb{R}^m$ . This is done by a series of  $(n+1)$  mappings  $\phi = \{\phi_0, \phi_1, \dots, \phi_n\}$  from  $\{\tilde{\mathbf{K}}_i\}$  to  $\{\mathbf{K}_i\}$  satisfying the following two conditions for any  $i \in [0, n]$  (cf. Mallet, 2002, 2.3.4):

- (i)  $\phi_i$  is a bijection between  $\tilde{\mathbf{K}}_i$  and  $\mathbf{K}_i$ ;
- (ii) if  $i > 0$ , then the bijection  $\phi_i$  and  $\phi_{i-1}$  preserve the incidence relationships:  $a' \in \partial a$  if and only if  $\phi_{i-1}(a') \in \partial \phi_i(a)$  for any “abstract”  $i$ -cell  $a \in \tilde{\mathbf{K}}_i$  and any of its incident  $(i-1)$ -cells  $a' \in \tilde{\mathbf{K}}_{i-1}$ .

Such bijections  $\phi$  are called an embedding of  $\{\tilde{\mathbf{K}}_i\}$  in the embedding space  $\mathbb{R}^m$  ( $m \geq n$ ). An example of an embedding is shown in Figure 3.11.



**Figure 3.11:** An general embedding model

Topology, represented by  $n$ -GMap, and embedding, represented by functions  $\phi$ , define geometric model ( $n$ -GModel).

**Definition 3.5** An  $n$ -GModel ( $n \geq 0$ ) is an algebra  $\mathcal{G}(D, A, \phi)$ , comprising the  $n$ -GMap  $\mathcal{M}(D, A)$  of a set of darts  $D$  and a set of involutions  $A = \{\alpha_0, \alpha_1, \dots, \alpha_n\}$ , and the embedding  $\phi$ .

Following Mallet (2002), linear geometric models can be defined as  $n$ -GModels comprising an embedding  $\phi$  constituted by  $\phi_0$  only. This is acceptable because we always construct embedding  $\phi$  from valid geometries. In this case, the condition to ensure the consistency of the map  $\varphi = \phi_0$  is

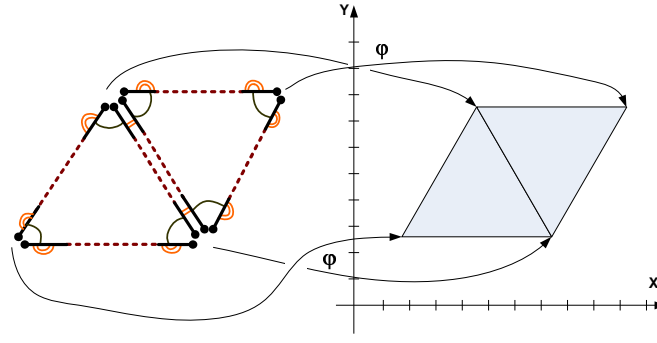
$$\varphi(d) = \varphi(d') \quad \text{for all } d' \in \langle \alpha_0 \rangle (d).$$

Figure 3.12 shows the embedding of a 2-GMap in 2-dimensional Euclidean space  $R^2$  using a linear embedding model.

### 3.4.2 Objects in spatio-temporal domain

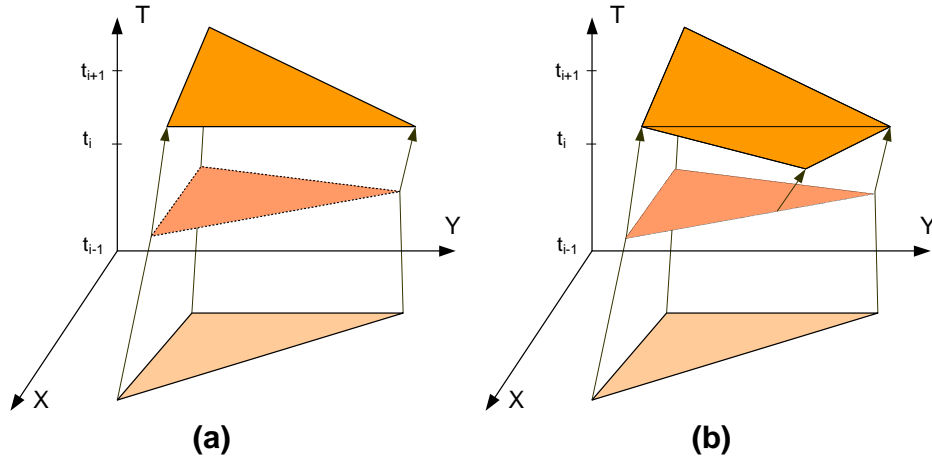
Objects in spatio-temporal domain can be represented as objects in  $m$ -dimensional Euclidean space  $R^m$  evolving along a time axis  $T$ . Two kinds of evolutions can be distinguished:





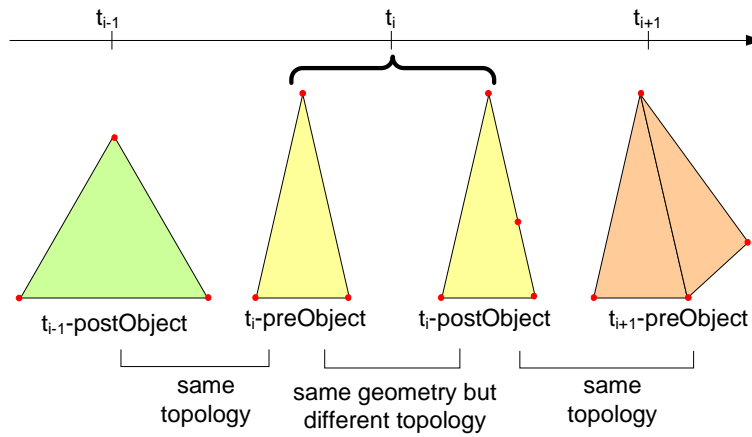
**Figure 3.12:** A linear embedding model

- evolutions which preserve all topological relationships concerning the objects (such in Figure 3.13a), i.e. homeomorphisms;
- evolutions which alter the topological relationships concerning the objects (such in Figure 3.13b).



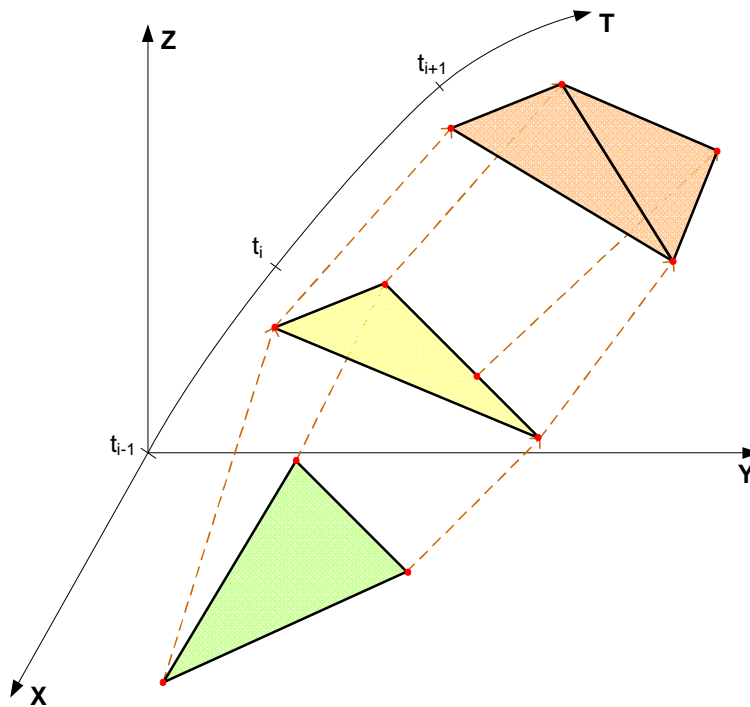
**Figure 3.13:** Geometry changes over time; (a) preserving the topology, (b) changing the topology

Following the suggestion by Polthier and Rumpf (1995), geo-objects are duplicated at each temporal instance into pre-object and post-object. They share a common geometry but differ in their topology. Any pre-object at a given instance has the same topology as the corresponding post-object of the previous instance, and any post-object at a given instance has the same topology as the pre-object of the next instance. For example, in Figure 3.14,  $t_i$ -pre-object is a triangle and has the same topology as  $t_{i-1}$ -post-object; the  $t_i$ -post-object comprises 2 triangles, where one is an ordinary and the other one is a degenerated triangle, and has the same topology as the  $t_{i+1}$ -pre-object; the  $t_i$ -pre-object and the  $t_i$ -post-object share a common geometry.



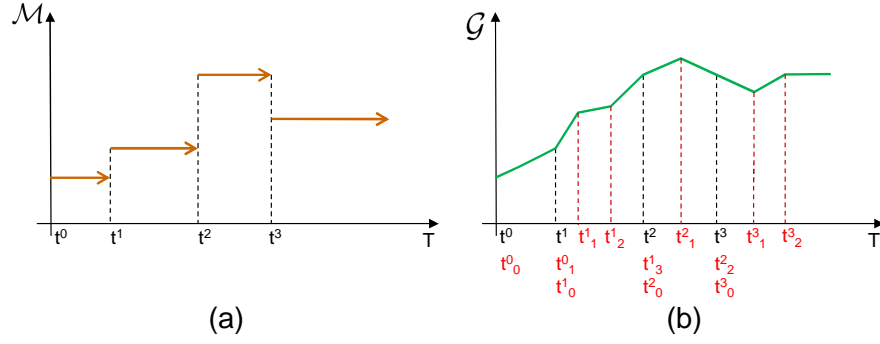
**Figure 3.14:** Topology synchronization

Figure 3.15 depicts the evolution of a geo-object in spatio-temporal domain. While the discontinuous evolution of topological relationships of an object can be



**Figure 3.15:** Example of an geo-object evolving in spatio-temporal domain

described by a step function as shown in Figure 3.16a, the geometrical evolution of an object can be described by a continuous piecewise linear function, where “sudden” changes are modeled by linear changes over a very short period of time as shown in Figure 3.16b.



**Figure 3.16:** (a) Topological function and (b) Geometrical function

Figure 3.17 provides a joint view of the topological and geometrical evolution of the objects in Figure 3.16a and Figure 3.16b. The topology of an object is a discrete function of  $t$  and can be described by a finite set of the topologies  $\mathcal{M}_T = \{\mathcal{M}(t_0), \mathcal{M}(t_1), \dots, \mathcal{M}(t_k)\}$  corresponding to the time series  $T^M = \{t_0, t_1, \dots, t_k\}$ . The geometry of an object is a function of time and by means of embedding of the object's topology. It can be described by a set of piecewise linear functions (in case of using linear interpolation) on the set of intervals  $\{(t_0, t_1), (t_1, t_2), \dots, (t_{k-1}, t_k)\}$  with the condition  $\mathcal{G}(\mathcal{M}(t_{i-1}), t_i) = \mathcal{G}(\mathcal{M}(t_i), t_i)$  for any  $0 < i \leq k$ . Each piecewise linear function  $\mathcal{G}(\mathcal{M}(t_i), t)$  on interval  $(t_i, t_{i+1})$ ,  $0 \leq i < k$ , is defined by the set of values

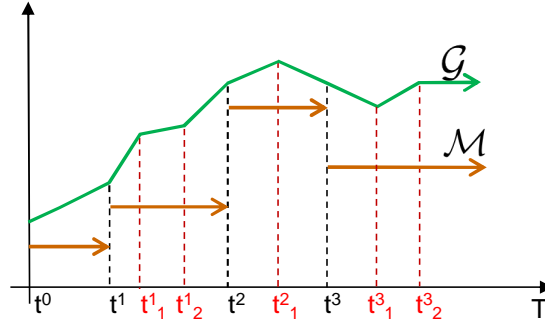
$$\{\mathcal{G}(\mathcal{M}(t_i), t_i^0), \mathcal{G}(\mathcal{M}(t_i), t_i^1), \dots, \mathcal{G}(\mathcal{M}(t_i), t_i^{g_i})\}$$

at times  $\{t_i = t_i^0, t_i^1, \dots, t_i^{g_i} = t_{i+1}\}$  and an interpolation method. If  $t > t_k$ , the geometry of the object is defined by the set of values

$$\{\mathcal{G}(\mathcal{M}(t_k), t_k^0), \mathcal{G}(\mathcal{M}(t_k), t_k^1), \dots, \mathcal{G}(\mathcal{M}(t_k), t_k^{g_k})\}$$

at times  $\{t_k = t_k^0, t_k^1, \dots, t_k^{g_k}\}$ . For convenience, if the object exists after time  $t_k^{g_k}$  then its geometry and topology are the same as at  $t_k^{g_k}$ , otherwise they are empty sets.

The calculation of the pre-object and the post-object from an object is the main task of a construction operation of the model which will be shown in Chapter 4.



**Figure 3.17:** Topology and Geometry over time

### 3.4.3 Geometric model in spatio-temporal domain

Let  $T$  be a collection of  $k + 1, k \geq 0$ , sequences, where the  $i^{th}$  sequence contains  $g_i + 1, g_i \geq 0$ , instances, and the initial instance of any sequence except the first equals the last instance of the previous sequence, i.e.,

$$T = \{\{t_0^0, t_0^1, \dots, t_0^{g_0}\}, \{t_1^0, \dots, t_1^{g_1}\} \dots, \{t_k^0, \dots, t_k^{g_k}\}\},$$

where  $t_{i+1}^0 = t_i^{g_i}$  for all  $0 \leq i < k$ . An example is  $T = \{\{1, 2, 3, 4\}, \{4, 5\}, \{5, 6, 7, 8, 9\}\}$ .

The set  $T^{\mathcal{M}}$  of initial instances of the sequences of  $T$  is

$$T^{\mathcal{M}} = \{t_0, t_1, \dots, t_k\} \text{ with } t_i = t_i^0 \text{ for all } i \in [0, k].$$

In the example,  $T^{\mathcal{M}} = \{1, 4, 5\}$ .

To generalize the geometric model in Section 3.4.1 from  $R^m$  into the domain  $R^m \times Time$ , we proceed as follows:

- (i) Expand the set of darts  $D$  to the set  $D_T$  such that  $D_T$  can be partitioned into  $k + 1$  non-empty and disjoint subsets  $D_i$  of darts, i.e.,

$$D_T = \bigcup_{i=0}^k D_i, D_i \neq \emptyset \text{ and } D_i \cap D_j = \emptyset \text{ for all } i \neq j.$$

- (ii) Expand the embedding function  $\phi$  into the collection of sequences, ordered and indexed according to  $T$  above, i.e.,

$$\Phi_T = \{\{\phi_0^0, \phi_0^1, \dots, \phi_0^{g_0}\}, \{\phi_1^0, \dots, \phi_1^{g_1}\} \dots, \{\phi_k^0, \dots, \phi_k^{g_k}\}\}.$$

The domain of  $\phi_i^0, \phi_i^1, \dots, \phi_i^{g_i}$  is  $D_i$ .

- (iii) Add constraints on the set  $\{\alpha_i\}$  such that  $\{\alpha_i\}$  is a set of involutions closed in each partition  $D_j$  of  $D_T$

$$\alpha_i(d_j) \in D_j, \quad \text{for all } d_j \in D_j.$$

Consequently, a new geometric model in the spatio-temporal domain, called  $n$ -TGSIS-Model, is accomplished and defined as follows.

**Definition 3.6** Let  $n, k$  be integers,  $n \geq 0, k \geq 0$ , and let  $g_i, i = 0, \dots, k$ , also be integers greater than or equal to 0. Let  $D_T, \Phi_T$  be defined as above, let  $A$  be a set of  $i$ -involutions for all  $i \in [0, n]$ . An  $n$ -TGSIS-Model is an algebra  $\mathcal{TGSIS}(D_T, A, \Phi_T)$  satisfying following constraints:

- (i)  $\alpha_i(d_j) \in D_j \quad \forall 0 \leq i \leq n; 0 \leq j \leq k; d_j \in D_j;$
- (ii)  $\alpha_i(d) \neq d \quad \forall d \in D_T; 0 \leq i < n;$
- (iii)  $\alpha_i \circ \alpha_{i+2+k} \{\alpha_i \circ \alpha_{i+2+k}(d)\} = d \quad \forall d \in D_T; i \geq 0; k \geq 0; i + 2 + k \leq n;$
- (iv)  $\alpha_i \circ \alpha_{i+2+k}(d) \neq d \quad \forall d \in D_T; i \geq 0; k \geq 0; i + 2 + k \leq n;$
- (v)  $\phi_i^j$  is an embedding  $\quad \forall 0 \leq i \leq k; 0 \leq j \leq g_i.$

The  $n$ -TGSIS-Model is equipped with an operation called “snapshot” to construct the geometry of an object at any given time instance  $t$ .

**Definition 3.7** Let  $\mathcal{TGSIS}(D_T, A, \Phi_T)$  be an  $n$ -TGSIS-Model, its snapshot at a given instance  $t$  is a  $n$ -GModel  $\mathcal{G}(D, A, \phi)$ , denoted by  $\mathcal{TGSIS}(t)$ , such that

- (i) if  $t_0 \leq t \leq t_k^{g_k}$ ,  $s$  be a variable whose value be defined by  $t$ , then
  - (1)  $D = \begin{cases} D_i & \text{if } t_i \leq t < t_{i+1} \rightarrow s = i \\ D_k & \text{if } t_k \leq t \leq t_k^{g_k} \rightarrow s = k, \end{cases}$
  - (2)  $\phi = \begin{cases} \phi_s^j + \frac{\phi_s^{j+1} - \phi_s^j}{t_s^{j+1} - t_s^j} \times (t - t_s^j) & \text{if } t_s^j \leq t < t_s^{j+1} \\ \phi_k^{g_k} & \text{if } t_k^{g_k} = t, \end{cases}$
- (ii) if  $t < t_0$ ,  $\mathcal{TGSIS}(t) = \emptyset$ ,
- (iii) if object exists when  $t > t_k^{g_k}$ , then  $\mathcal{TGSIS}(t) = \mathcal{TGSIS}(t_k^{g_k})$  for  $t > t_k^{g_k}$ . If not,  $\mathcal{TGSIS}(t) = \emptyset$  for  $t > t_k^{g_k}$ .

### 3.4.4 The model in object–relational form

In this section,  $n$ –TGSIS–Model is converted into object–relational form. To this end, the following tasks are done.

First, the collection of sequences  $T = \{\{t_0^0, t_0^1, \dots, t_0^{g_0}\}, \{t_1^0, \dots, t_1^{g_1}\}, \dots, \{t_k^0, \dots, t_k^{g_k}\}\}$  is represented as two relations *TimeT* and *TimeM* as follows. The relation *TimeT* represents the sequence of time instances at which the geometry of the object has changed  $T^T = \{t_0^0, t_0^1, \dots, t_0^{g_0-1}, t_1^0, \dots, t_1^{g_1-1}, \dots, t_k^0, \dots, t_k^{g_k}\}$ , and *TimeM* represents the sequence of time instances at which the topology of the object has changed  $T^M = \{t_0, t_1, \dots, t_k\}$ . For example, if  $T = \{\{1, 2, 3, 4\}, \{4, 5\}, \{5, 6, 7, 8, 9\}\}$ , then *TimeT* contains values  $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$  and *TimeM* contains values  $\{1, 4, 5\}$ . Second, the set  $D_T = \{D_{t_0}, D_{t_1}, \dots, D_{t_k}\}$  is represented as the relation *Dart* with an attribute *TimeMID* in  $[0, k]$ . Next, the set  $A = \{\alpha_0, \alpha_1, \dots, \alpha_n\}$  of involutions is represented as the relation *Alpha* with attributes *level* in  $[0, n]$ , *DartID1*, and *DartID2* to represent the map from dart  $d_1$  to dart  $d_2$ . Finally, the set  $\Phi = \{\{\phi_0^0, \phi_0^1, \dots, \phi_0^{g_0}\}, \{\phi_1^0, \dots, \phi_1^{g_1}\}, \{\phi_k^0, \dots, \phi_k^{g_k}\}\}$  is represented as the relation *Phi* with attributes *TimeID*, *DartID*, *CoordinateID* to map sets of darts, e.g. 0-orbits, to geometrical cells in real space  $\mathbb{R}^m$ .

Figure 3.18 shows the  $n$ –TGSIS–Model in object–relational mode, referred to as  $n - TGSIS - Model\ in\ OR$ .

The attribute *Alive* (boolean attribute) in the relation *Feature* is to define the object existing after the point of time  $t_k^{g_k}$  or not.

To reduce the number of join operations, and therefore improve the performance of geometric queries, a relation *Node* representing abstract 0–cells is added, and the relations *Coordinate*, and *Phi* are merged into a relation named *Vertex*. The Figure 3.19 depicts  $n$ –TGSIS–Model in OR with redundant relation *Node*.

### 3.4.5 Assigning geoscience properties to geometry

Almost all geoscience applications need to assign geological, geophysical, geochemical properties to parts of geometric objects to solve problems. For example, exploration and production applications in petroleum fields usually require properties such as porosity, permeability, etc. Therefore, besides modeling geometric objects evolving in space and time, properties need to be assigned to geometries

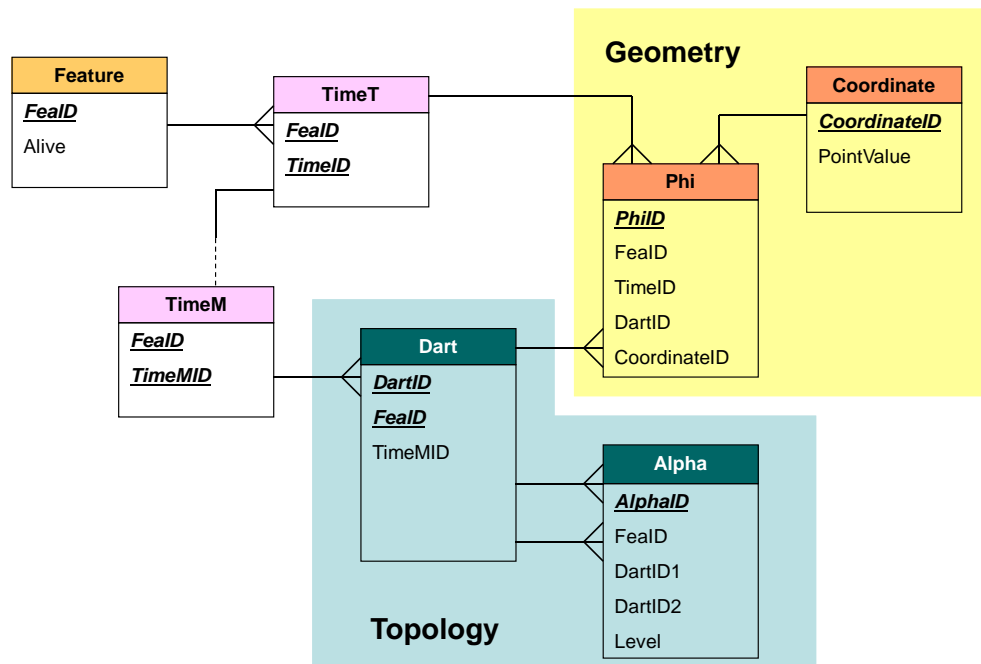


Figure 3.18: Object-relational model of TGSIS

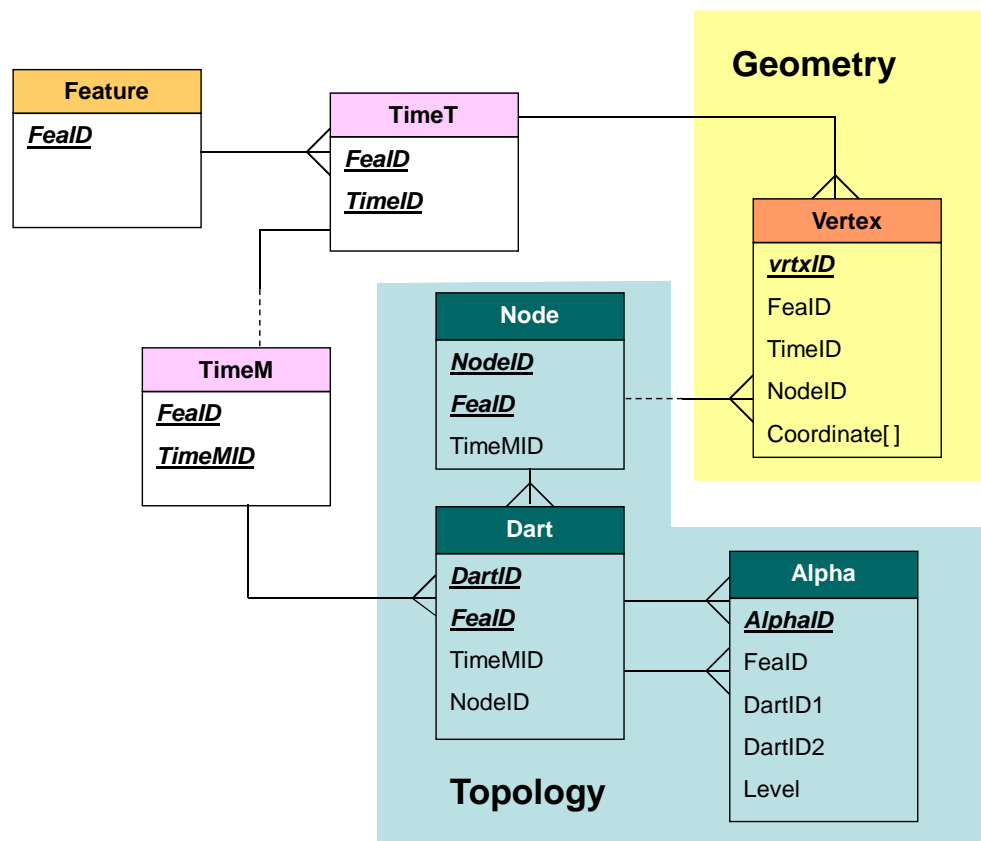
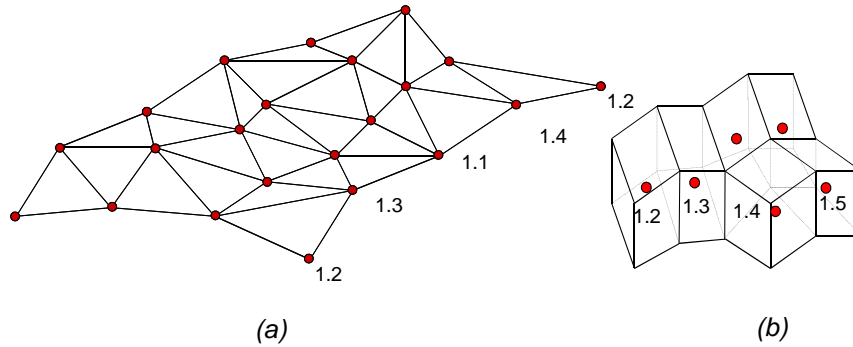


Figure 3.19: Object-relational model of TGSIS with redundant relation Node

and they also evolve in time. Properties are assigned to cells, usually either vertices or volumes, of geo-objects. Figure 3.20 shows an example of two geometric objects with properties assigned either to the vertices (Fig. 3.20a) or to the 3-cells (Fig. 3.20b).



**Figure 3.20:** Porosity values are assigned either to vertices of a horizontal surface (a), or to 3-cells of a geo-object (b)

Users of the TGSIS-Model decide where to archive geo-objects and which properties should be assigned, i.e., user-defined relations in the database. These relations and their attributes are designed by database administrators and stored in user schema. Relation *Catalog* and a stored procedure called *TGSIS\_REGISTER* are added to the model to register and keep information connecting between user schema and internal TGSIS schema. Figure 3.21 shows a TGSIS-Model with properties.

The “User schema” (see Fig. 3.21) comprises user-defined relations including  $\langle FeatureLayer \rangle$  and  $\langle Properties \rangle$ . For example, in structural geology, relation *STRUCTURAL\_HORIZONTAL* is designed to store horizontal surfaces with attributes ID for identifier, SHAPE for geometry of surface, and NAME for name of surface. Properties POROSITY and PERMEABILITY are stored in relation *STRUCTURAL\_HOR\_PROS*, and assigned to the vertices of these surfaces. The stored procedure *TGSIS\_REGISTER* is used to write register information about user schema to the *Catalog* relation. The example is described in SQL language using PostgreSQL as

- Create table *STRUCTURAL\_HORIZONTAL*(ID bigint, SHAPE bigint, NAME varchar(50));
- Create table *STRUCTURAL\_HOR\_PROS*(POROSITY float, PERMEABILITY float);



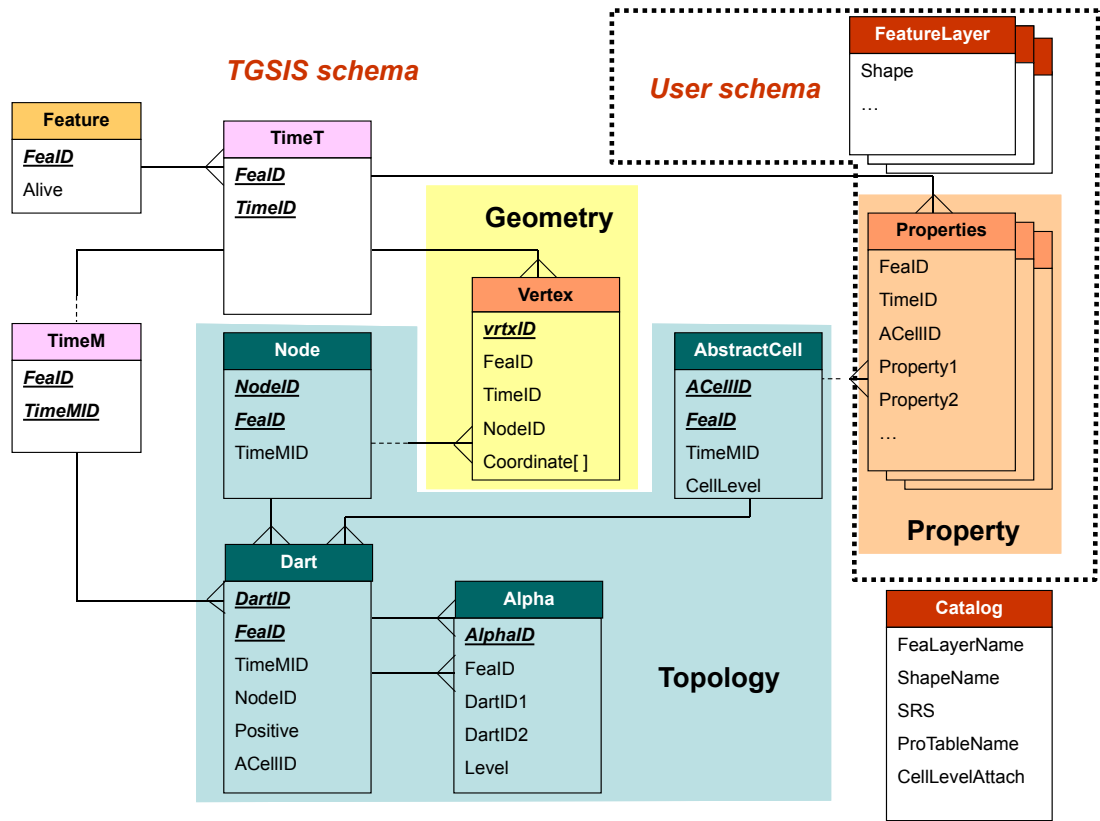


Figure 3.21: Object-relational model of TGSIS with properties

- *Select TGSIS\_REGISTER('STRUCTURAL\_HORIZONTAL', 'SHAPE', 0, 'STRUCTURAL\_HOR\_PROS', 0);*

Three attributes *FealID*, *TimeID*, *AcclIID* will be added into relation *STRUCTURAL\_HOR\_PROS* by procedure *TGSIS\_REGISTER*. The TGSIS schema comprises internal relations that cannot be changed directly by users. Relation *AbstractCell* is added to reduce the number of join operations in property queries. A node is an abstract cell with dimension (*CellLevel*) 0, which always assigns to coordinates, therefore relation *Node* is kept to get high performance of geometric query.

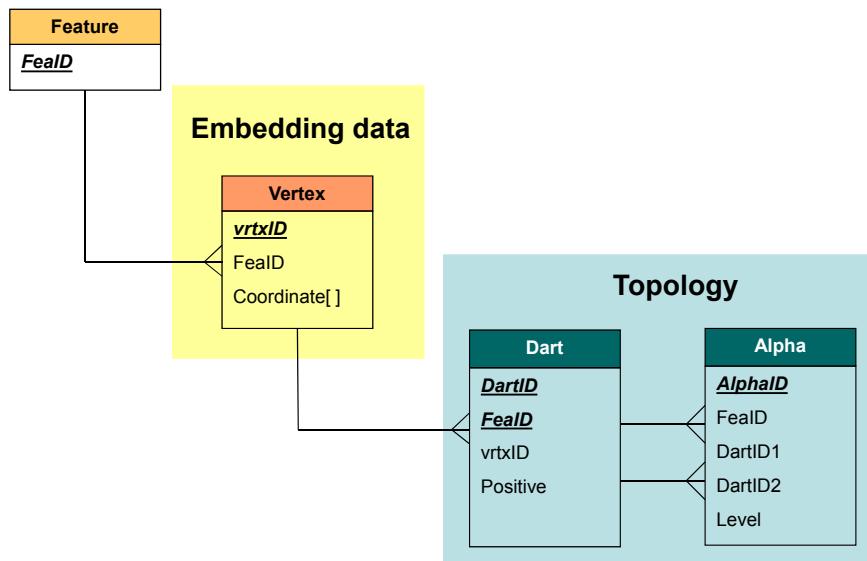
### 3.5 A model for multiple indexed geoscience data

In the more general approach to integrate time into space, one- or multi-dimensional time is considered as additional spatial dimensions. This approach was used by Worboys (1994) and, more recently, in the research by Peter van Oosterom's group

(Ohori et al., 2013b, van Oosterom and Stoter, 2010). This approach forms larger dimensional objects. For example, a polygon in a two-dimensional space moving in time is considered as a polyhedron in a three-dimensional space (two spatial dimensions and one time dimension). The approach is extensible and generic because it treats objects in a dimension independent manner. Additional spatial dimensions can be interpreted as time, scale (Ohori et al., 2013b, van Oosterom and Stoter, 2010), uncertainty (Caumon, 2010), or any non-spatial properties.

While the approach is theoretically sound, its implementation is problematic since it requires an appropriate data structure, and methods for data modeling, querying, and analysing. Ohori et al. (2013b) proposed the use of generalised maps (GMaps) to represent larger-dimensional GIS datasets because GMaps are able to represent a wide class of objects in arbitrary dimensions (see Section 3.4.1). A method to model larger dimensional data, i.e., extrusion, was proposed by Ohori and Ledoux (2013). Certain techniques that are necessary in order to extract meaningful 2D/3D information from larger dimensional data were presented in Ohori et al. (2013a).

Using the method given by Ohori et al. (2013b), we model larger dimensional geo-objects using generalised maps (Lienhardt, 1989, 1994). The data schema is shown in Figure 3.22.



**Figure 3.22:** A data schema of the model for multiply indexed geoscience data

## 3.6 Summary

This chapter has presented five data models for geoscience information systems. Two of the models represent geological surfaces existing in 3-dimensional space, evolving in time discretely or continuously. These two data models have been implemented in the PostgreSQL database management system (PostgreSQL, 2014) to build the prototype software presented in Chapter 6. One data model represents geological objects with the geometry, topology, and physical properties in the 3-dimensional space. A similar data model has been implemented in the GST-Framework (Gabriel et al., 2012, GiGa-Infosystems, 2014). The TGSIS data model, which is a general data model for objects in the domain  $R^m \times time$ , is reviewed in this chapter. The last model was proposed for multiple indexed geoscience data in which time and other non-spatial dimensions are interpreted as larger spatial dimensions.

Choosing which data model to be implemented is dependent on types of data. When the application is only required to manage static 3D objects, the first data model should be chosen. To manage surfaces changing in time discretely or continuously, the second or third data model should be used, respectively. The fourth data model should be used when the application is required to manage objects with larger spatial dimensional geometries evolving continuously in one valid time dimension. To manage objects in multiple time dimensions, the fifth data model should be used. Each data model includes a data structure and operations, algorithms on this data structure. For the fourth data model, only an operation, i.e., “snapshot” is described. Other operations and algorithms is the subject of the future work. For the fifth data model, all operations and algorithms are also left to the future work. Many operations and algorithms on the first three data models may immediately be inherited from geomodeling systems, such as gOcad.



# Chapter 4:

## Morphological Interpolation

---

### 4.1 Introduction

Morphology is the study of the shapes or the forms of objects. In this study, the term morphological interpolation refers to calculating the intermediate shapes between two or more known shapes. Morphological interpolation is primarily used to construct a 3-dimensional object from its sparse 2-dimensional cross-sections for visualisation or analysis. For example, in tomographic medical imaging, morphological interpolation is used to reconstruct an organ or other object of interest, such as the heart or a tissue, from a series of 2-dimensional slices; in the geosciences, morphological interpolation provides useful tools for the 3-dimensional modeling, visualisation, and analysis of geological bodies. Moreover, morphological interpolation can be used for morphing, decomposition, and many other purposes.

In the context of a spatio-temporal geoscience information system, a geo-object can be considered as it exists in the 3-dimensional Euclidean space and evolves along the real time axis. The evolution of the geo-object is caused by certain geological processes, such as deposition, erosion, or uplift. To correctly represent this evolution, physical process models are required. Unfortunately, in many cases, the physical process models are unavailable or there are insufficient data to feed these models. The available data merely represent the object in the 3-dimensional Euclidean space at certain time instances. These data, which include

the geometry, topology, and properties, can be considered as 3-dimensional cross-sections of a 4-dimensional object. Therefore, morphological interpolation can aid in the construction of this object. In summary, the issue is stated as follows:

*Given two data models (the triangulated mesh model or the voxel model), find intermediate models that are as similar as possible in shape and size to the given models.*

Mathematical morphology is a theory and technique used for the analysis and processing of geometrical structures (Heimans and Ronse, 1990, Serra, 1983). Mathematical morphology is most commonly applied to multidimensional images. Currently, there are several studies on the application of mathematical morphology to graphs and simplicial complexes (Cousty et al., 2009, Dias et al., 2011, 2014). However, these studies are aimed at manipulating the values associated with the elements of the graph or the simplicial complex but do not change the shapes of these structures. In the case of representing geo-objects by triangulated mesh surfaces (a type of simplicial complex), mathematical morphology remains useful when the following procedure is applied. First, the triangulated mesh surfaces are converted to 3-dimensional images (voxels). Second, mathematical morphology techniques are applied to these images. Finally, the processed images are converted back to mesh surfaces.

In addition, many mesh processing techniques have been invented to directly manipulate meshes (Botsch et al., 2010, Hormann et al., 2008). For morphological interpolation, the related techniques are mesh parameterisation, cross-parameterisation, compatible remeshing, and mesh morphing. A morphological interpolation using mesh processing techniques typically contains two steps. The first step searches for a bijective map between the source mesh and the target mesh, known as the vertex correspondence or the consistent/compatible meshes. The second step chooses a set of trajectories along which the corresponding vertices travel as they evolve from the first set of vertices into the second set of vertices. This process is known as vertex path/trajectory (Alexa, 2000, Floater and Gotsman, 1999, Liu et al., 2011, Yan et al., 2007). Regarding the first step, van Kaick et al. (2011) provided a recent review of compatible triangulated meshes. The compatible meshes are typically computed by subdividing two meshes into their corresponding patches, which are homeomorphic to disks. Second, the parameterisation of the two meshes on a common base domain is calculated. Third, the cross-parameterisation is calculated, and finally, the meshes are remeshed

(Kraevoy and Sheffer, 2004, Praun et al., 2001).

In the scope of this thesis, we developed a new morphological interpolation algorithm based on mesh processing techniques, tailored for the construction of geo-objects in 4-dimensional space (Le, 2013). This algorithm includes the following four main procedures: (1) cutting the surfaces, (2) setting up the constraints, (3) partitioning, and (4) calculating the parameterisations and trajectories. The algorithm especially functions with the geometry constraints in which a surface always attaches to other surfaces (control surfaces) during its evolution.

This chapter is organised as follows. Section 4.2 reviews the mathematical morphological theory and the morphological interpolation methods in the framework of the mathematical morphology. In Section 4.3, the concepts and techniques of mesh processing are presented. Our morphological interpolation algorithm is reviewed in Section 4.4. Finally, Section 4.5 is devoted to the summary.

## 4.2 Interpolation using mathematical morphology

### 4.2.1 Mathematical morphology

Mathematical morphology was invented in 1964 by Georges Matheron and Jean Serra (Serra, 1983). Mathematical morphology typically address the mathematical theory of describing shapes using sets. Mathematical morphology provides a powerful approach to processing images including filtering, segmentation and interpolation. Heimans and Ronse (1990) extended mathematical morphology using complete lattices. Since then, this approach has been applied to more complex digital structures, such as graphs, hyper-graphs, and simplicial complexes. In the remainder of this section, only mathematical morphology on the sets (multi-dimensional binary images) is presented.

The basic idea of mathematical morphology is the use of a pre-defined shape, called structuring element, to probe an image to draw conclusions on whether this shape fits the shapes in the image. A structuring element itself is also a binary image. The two basis morphological operations are dilation and erosion. The sequential combination of these two operations gives rise to more complex operations. Denote the *translation* of the set  $X$  by the vector  $t$  as  $X_t = \{p | p - t \in X\}$  and the *point-inversion* of a set  $B$  as  $\check{B} = \{-b | b \in B\}$ . The *dilation*  $\delta(X)$  of

a set (binary image)  $X$  by the structuring element  $B$  is defined by the formula

$$\delta_B(X) = \{p | B_p \cap X \neq \emptyset\} \quad (4.1)$$

or

$$\delta_B(X) = \bigcup_{b \in B} X_{-b}. \quad (4.2)$$

The erosion  $\varepsilon(X)$  of a set  $X$  by a structuring element  $S$  is defined by the formula

$$\varepsilon_B(X) = \{p | B_p \in X\} \quad (4.3)$$

or

$$\varepsilon_B(X) = \bigcap_{b \in B} X_{-b}. \quad (4.4)$$

Using Minkowski sum,  $\bigcup_{b \in B} X_b$ , and Minkowski difference,  $\bigcap_{b \in B} X_B$ , dilation and erosion are

$$\delta_B(X) = X \oplus \check{B}$$

and

$$\varepsilon_B(X) = X \ominus \check{B}$$

Thus, if the structuring element  $B$  is symmetrical, i.e.  $B = \check{B}$ , dilation and erosion are equivalent to Minkowski sum and Minkowski difference, respectively.

## 4.2.2 Morphological interpolation in the framework of mathematical morphology

Iwanowski (2014) presented certain interpolation algorithms that were developed by J. Serra (Serra, 1994, 1998), F.Meyer (Meyer, 1994a,b), and S. Beucher (Beucher, 1994, 1998).

### Morphological interpolation by the Hausdorff distance

J. Serra (Serra, 1994, 1998) used the notation of the Hausdorff distance, defined as follows. The Hausdorff distance between two sets  $X$  and  $Y$  is defined according to the following equation:

$$\rho(X, Y) = \max\{\sup_{x \in X} d(x, Y), \sup_{y \in Y} d(y, X)\} \quad (4.5)$$



where  $d(x, Y) = \inf\{d(x, y) : y \in Y\}$ . The Hausdorff distance can also be defined according to the following equation:

$$\rho(X, Y) = \inf\{\lambda : X \subseteq \delta_\lambda(Y); Y \subseteq \delta_\lambda(X)\} \quad (4.6)$$

where  $\delta_\lambda(X)$  is the dilation of  $X$  by the compact ball centred at the origin and the radius  $\lambda$ . The first Hausdorff geodesic interpolation between set  $X$  (initial image) and set  $Y$  (final image) is given by the following equation:

$$Z_\alpha = \delta_{\alpha\rho}(X) \cap \delta_{(1-\alpha)\rho}(Y), \alpha \in [0, 1], \quad (4.7)$$

where  $\rho$  is the Hausdorff distance between  $X$  and  $Y$  and coefficient  $\alpha$  determines the relative position of the interpolated set  $Z_\alpha$ .

The second Hausdorff geodesic interpolation is restricted only to the case in which both of the sets  $X$  and  $Y$  are convex. The formula  $(1 - \alpha)X \oplus \alpha Y$  indicate that the dilation of set  $X$  is reduced by the factor  $(1 - \alpha)$  with the structuring element equal to set  $Y$  reduced by the factor  $\alpha$ . The interpolation can be expressed as follows:

$$Z_\alpha = \delta_{\alpha\rho}(X) \cap \delta_{(1-\alpha)\rho}(Y) \cap (1 - \alpha)X \oplus \alpha Y, \alpha \in [0, 1]. \quad (4.8)$$

To avoid excessively large sizes of interpolated sets, the Hausdorff distance between the sets can be reduced. This reduction can be performed by moving the sets to a common position using a displacement vector. Then, the interpolated set is produced. Finally, the interpolated set is moved to its proper position using the displacement vector in the first step.

## Morphological median

The morphological median of two sets is a new set (the median set) that has a shape midway between the shapes of the two initial sets. In this subsection, we use the notation of the *influence zones* of the sets. If  $P_1, P_2, \dots, P_n$  are disjointed sets, then the influence zone of  $P_i$  is the locus of those points that are closer to  $P_i$  than to any of the other sets. If only two sets are involved, then  $P$  and  $Q$  ( $Q \subset P$ ), the influence zone of  $Q$  with respect to the complement of  $P$ ,  $P^c$ , is still

called the influence zone of  $Q$  inside  $P$ , according to the following expression:

$$IZ_P(Q) = \{x : d(x, Q) < d(x, P^C)\}. \quad (4.9)$$

The influence zone defined by Equation 4.9 represents a median set between the two initial sets, one included in the another, according to the following equation:

$$M(P, Q) = IZ_P(Q). \quad (4.10)$$

The median set also satisfies the following equation:

$$M(P, Q) = \bigcup_{\forall \lambda} \{(Q \oplus \lambda B) \cap (P \ominus \lambda B)\}, \quad (4.11)$$

where  $\oplus \lambda B$  is the dilation of size  $\lambda$  and  $\ominus \lambda B$  represents the erosion of size  $\lambda$ , both with the elementary structuring element  $B$ .

In the case of two sets with a non-empty intersection ( $X \cap Y \neq \emptyset$ ), the median set of  $X$  and  $Y$  is defined by the influence zone of  $X \cap Y$  in  $X \cup Y$ , according to the following equation:

$$M(X, Y) = IZ_{(X \cup Y)}(X \cap Y) = \bigcup_{\forall \lambda} \{((X \cap Y) \oplus \lambda B) \cap ((X \cup Y) \ominus \lambda B)\}. \quad (4.12)$$

In the Equation 4.12, the median set  $M(X, Y)$  can be calculated using the following iterative procedure.

First, three auxiliary sets,  $Z$ ,  $W$ , and  $M$  are assigned as follows:

$$\begin{aligned} Z_0 &= X \cap Y, \\ W_0 &= X \cup Y, \\ M_0 &= X \cap Y. \end{aligned} \quad (4.13)$$

New values of  $Z_i$ ,  $W_i$ , and  $M_i$  are computed iteratively, as follows:

$$\begin{aligned} Z_i &= Z_{i-1} \oplus B, \\ W_i &= W_{i-1} \ominus B, \\ M_i &= (Z_i \cap W_i) \cup M_{i-1}. \end{aligned} \quad (4.14)$$

The iterations are performed until idempotence, i.e.,  $M_i = M_{i+1}$ .

### Interpolation function

The interpolation function method was introduced by F. Meyer (Meyer, 1994a,b). The interpolation function is a distance function. The method requires that the two initial sets have a non-empty intersection. This requirement can be released by moving both of the sets into a common position.

Let  $X$  and  $Y$  be the initial and final sets, respectively. The interpolated image between the sets should be generated as a union of the appropriate intermediary set between  $X$  and  $X \cap Y$  and the appropriate intermediary set between  $X \cap Y$  and  $Y$ . The interpolation function between the set  $U = X \cap Y$  and  $V = Y$  is constructed as follows.

First, the geodesic distance  $d_1$  to  $V^C$  in  $U^C$  is calculated. The level lines of this distance can be obtained by successive geodesic erosions of  $V$  in  $U^C$ . Second, the geodesic distance  $d_2$  to  $U$  in  $V$  is calculated by the successive geodesic dilations of  $U$  in  $V$ . The final interpolation function is generated using the following equation:

$$d = \frac{d_1}{d_1 + d_2}. \quad (4.15)$$

An appropriate interpolator can be defined as follows:

$$Int_U^V(\alpha) = \{x : d(x) \leq \alpha\}, \quad \alpha \in [0, 1]. \quad (4.16)$$

To obtain the interpolated set between the two sets  $X$  and  $Y$  (with a non-empty intersection) at level  $\alpha$ , we threshold the interpolation function between  $Y \cap X$  and  $X$  at level  $\alpha$  and threshold the interpolation function between  $X \cap Y$  and  $Y$  at level  $(1 - \alpha)$ . The final interpolated set is obtained by taking the union of the results of both of the thresholded distance functions, as follows:

$$Interpol_X^Y(\alpha) = Int_{(X \cap Y)}^X(\alpha) \cup Int_{(X \cap Y)}^Y(1 - \alpha), \quad \alpha \in [0, 1]. \quad (4.17)$$

### Morphological shape-based interpolation

Bors et al. (2002) proposed the morphological shape-base interpolation method to reconstruct the  $n$ -dimensional object from a group of  $(n-1)$ -dimensional sets representing sections of that object. Let  $P$  and  $Q$  be two sets representing two shapes in an  $n$ -dimensional space denoted as  $E$ . The goal of this method is to

construct a sequence of sets showing a gradual transition between the two given shapes.

The transformation influences the elements located on the boundary of the set  $P$ , as follows:

$$C_P = \{c : c \in P, \exists c_1 \in P^C, c_1 \in N_B(c)\} \quad (4.18)$$

where  $N_B(c)$  denotes the neighbourhood of the element  $c$  that has the same size and shape as the structuring element  $B$ . In the morphing operation, the element of a boundary set  $C_P$  changes differently according to its correspondence with the other given set  $Q$ . This change is defined by a sequence of dilation and erosion operations. Three possible correspondence cases are defined as follows.

The first case occurs when the border region of one set corresponds to the interior of the other set. In this case, dilations are applied to the border elements, as follows:

$$\begin{aligned} &\text{If } p_m \in C_P \wedge q_m \in Q - C_Q \\ &\text{then perform } p_m \oplus B_1 \end{aligned}$$

where  $B_1$  is the structuring element. The second case occurs when the border region of one set corresponds to the background of the other set. In this case, erosions of the boundary elements are performed as follows:

$$\begin{aligned} &\text{If } p_m \in C_P \wedge q_m \in Q^C \\ &\text{then perform } p_m \ominus B_1. \end{aligned}$$

In the last case, both of the corresponding elements are members of their set boundaries, and no change is required.

$$\begin{aligned} &\text{If } p_m \in C_P \wedge q_m \in C_Q \\ &\text{then perform no change.} \end{aligned}$$

The morphing transformation applied to set  $P$  depending on set  $Q$  and on the structuring element  $B_1$  is defined as follows:

$$f(P|Q, B_1) = [(P \ominus B_1) \cup ((P \cap Q) \oplus B_1)] \cap (P \cup Q). \quad (4.19)$$

Similarly, the morphing operation is defined on set  $Q$  depending on set  $P$  and on

the structuring element  $B_2$ :

$$f(Q|P, B_2) = [(Q \ominus B_2) \cup ((P \cap Q) \oplus B_2)] \cap (P \cup Q). \quad (4.20)$$

**Theorem 4.1** *We can always generate an intermediary set between two sets  $P$  and  $Q$ , satisfying  $P \cap Q \neq \emptyset$ , by iterating the set transformations defined by two equals onto their previous iteration output sets until idempotency.*

Theorem 4.1 was proven by Bors et al. (2002). Using this theorem, the median set of two initial sets can be constructed using the same structuring element for both of the above equations.

## 4.3 Interpolation using mesh parameterisation

### 4.3.1 Mesh parameterisation

The parametric representations of a surface  $S$  are defined by a vector-value parameterisation function  $f : \Omega \rightarrow R^3$ , where  $\Omega \subset R^2$  is the parameter domain. In many applications, the function  $f$  and the parameter domain  $\Omega$  are unknown, but the surface  $S$  is given, primarily in triangulated or polygonal mesh forms. Finding a function  $f$  and  $\Omega$  (in certain cases) from a given  $S$  is referred to as parameterisation. The initial application of parameterisation found in computer graphics is mapping of textures onto surfaces. Beyond this application, parameterisation is currently used for many mesh processing applications, including detail-mapping, detail-transfer, morphing, mesh-editing, mesh-completion, remeshing, compression, surface-fitting, and shape-analysis (Botsch et al., 2010, Hormann et al., 2008).

Assume that we are given a triangulated mesh  $S$  that is a set of triangles  $\mathcal{T} = \{T_1, T_2, \dots, T_m\}$  that intersect only at common edges  $\mathcal{E} = \{E_1, E_2, \dots, E_l\}$  and vertices  $\mathcal{V} = \{p_1, p_2, \dots, p_n, \dots, p_{n+b}\}$ . More specifically, the set of vertices consists of  $n$  interior vertices  $\mathcal{V}_I = \{p_1, p_2, \dots, p_n\}$  and  $b$  boundary vertices  $\mathcal{V}_B = \{p_{n+1}, p_{n+2}, \dots, p_{n+b}\}$ . Two distinct vertices  $p_i, p_j \in \mathcal{V}$  are called neighbours if they are the end points of an edge  $E = [p_i, p_j] \in \mathcal{E}$ , and for any  $p_i \in \mathcal{V}$ , we denote  $N_i = \{j : [p_i, p_j] \in \mathcal{E}\}$  as the set of indices of all of the neighbours of  $p_i$ . Because  $S$  is known and  $\Omega$  is unknown, parameterisation methods typically find

the inverse of  $f$ , denoted by  $g = f^{-1}$ . Let us review the simplest case in which  $S$  is a triangle defined by three distinct points in  $R^3$ ,  $T(p_1, p_2, p_3)$ , and  $\Omega$  is also a triangle defined by three distinct points in  $R^2$ ,  $t(u_1, u_2, u_3)$ . Every point  $p$  in  $S$  can be written in a unique formula as a barycentric combination of the corner points, as follows:

$$p = \alpha p_1 + \beta p_2 + \gamma p_3, \quad (4.21)$$

where  $\alpha + \beta + \gamma = 1, \alpha, \beta, \gamma \geq 0$ .

We can now define a linear mapping  $f : \Omega \rightarrow S$  with the following:

$$\alpha u_1 + \beta u_2 + \gamma u_3 \rightarrow \alpha p_1 + \beta p_2 + \gamma p_3.$$

It is simple to prove that the function  $f$  is a bijective function from triangle  $t$  in 2D space to triangle  $T$  in 3D space. Hence, the inverse function of  $f$ ,  $g = f^{-1}$ , is defined.

From the above review, we can conclude that mapping  $g$  is uniquely determined by specifying the parameter points  $u_i = g(p_i)$  for each vertex  $p_i \in V$  and demanding that  $g$  is continuous and linear for each triangle. Specifically,  $g_T$  is the linear map from a surface triangle  $T[p_i, p_j, p_k]$  to the corresponding parameter triangle  $t[u_i, u_j, u_k]$ , and  $f_t = (g_T)^{-1}$  is the inverse linear map from  $t$  to  $T$ . The parameter domain  $\Omega$  is the union of all of the parameter triangles. To have a valid parameterisation  $f$ , which is a piecewise linear function defined by a set of  $f_t$ , it is required that the parameter domain  $\Omega$  does not self-intersect, indicating that the intersection of any two triangles in the parameter space is either a common edge, a common vertex, or empty.

Briefly, the goal of mesh parameterisation is to find a set of coordinates  $(u_i, v_i)$  associated with each vertex  $p_i$  of the mesh such that the image of this mesh in the parameter space does not self-intersect.

### Parameterisations based on barycentric coordinates

Constructing a parameterisation of a triangle mesh can be performed based on the following physical model. It is imagined that the edges of the triangle mesh are ideal springs, i.e., the remaining length is zero, and the potential energy is  $\frac{1}{2}Ds^2$ , where  $D$  is the spring constant and  $s$  is the length of the spring. Then, the boundary of this spring network is glued somewhere in the plane so the interior of

this network relaxes with the most efficient energy of the configuration. Finally, we simply assign the positions where the joints of the network have come to as parameter points.

The overall spring energy is determined by the following formula:

$$E = \frac{1}{2} \sum_{i=1}^{n+b} \sum_{j \in N_i} \frac{1}{2} D_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|^2. \quad (4.22)$$

In this formula,  $D_{ij} = D_{ji}$  is the spring constant of the spring between  $\mathbf{p}_i$  and  $\mathbf{p}_j$ , the parameter positions  $\mathbf{u}_i = (u_i, v_i)$  are known for all  $i = n+1, \dots, n+b$ , i.e., the boundary points and are unknown for all  $i = 1, \dots, n$ , i.e., the interior points.

The minimum value of  $E$  is obtained when the following expression is true:

$$\frac{\partial E}{\partial \mathbf{u}_i} = 0, \text{ i.e., } \sum_{j \in N_i} D_{ij} (\mathbf{u}_i - \mathbf{u}_j) = 0 \text{ or } \mathbf{u}_i \sum_{j \in N_i} D_{ij} = \sum_{j \in N_i} D_{ij} \mathbf{u}_j \quad (4.23)$$

for all  $i = 1, \dots, n$ . This equation shows that each interior parameter point  $\mathbf{u}_i$  is an affine combination of its neighbours, i.e., according to the following expression:

$$\mathbf{u}_i = \sum_{j \in N_i} \lambda_{ij} \mathbf{u}_j, \quad (4.24)$$

with normalised coefficients according to the following expression:

$$\lambda_{ij} = D_{ij} / \sum_{k \in N_i} D_{ik}$$

that sums to 1.

Separating the known and the unknown parameter points on the right hand side of Equation 4.24 we have the following equation:

$$\mathbf{u}_i - \sum_{j \in N_i, j \leq n} \lambda_{ij} \mathbf{u}_j = \sum_{j \in N_i, j > n} \lambda_{ij} \mathbf{u}_j. \quad (4.25)$$

Computing the coordinates  $u_i$  and  $v_i$  of the interior parameter points  $\mathbf{u}_i(u_i, v_i)$  requires solving the two following linear systems of equations:

$$AU = \bar{U} \quad \text{and} \quad AV = \bar{V}. \quad (4.26)$$

In Equations 4.26,  $U = (u_1, \dots, u_n)$  and  $V = (v_1, \dots, v_n)$  are the column vectors of the unknown coordinates, and  $\bar{U} = (\bar{u}_1, \dots, \bar{u}_n)$  and  $\bar{V} = (\bar{v}_1, \dots, \bar{v}_n)$  are the column vectors with the following coefficients:

$$\bar{u}_i = \sum_{j \in N_i, j > n} \lambda_{ij} u_j \quad \text{and} \quad \bar{v}_i = \sum_{j \in N_i, j > n} \lambda_{ij} v_j. \quad (4.27)$$

The matrix  $A = (a_{ij})_{i,j=1,\dots,n}$  is the  $n \times n$  matrix with the following elements:

$$a_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -\lambda_{ij} & \text{if } j \in N_i, \\ 0 & \text{otherwise.} \end{cases}$$

The two sparse linear systems of equations 4.26 can be solved efficiently by solvers, including OpenNL (OpenNL, 2014), Eigen (Guennebaud et al., 2010), and TAUCS (Toledo et al., 2014).

The remaining question is to determine the spring constants  $D_{ij}$  in the spring model or to determine the normalised coefficients  $\lambda_{ij}$  in Equation 4.24. W. T. Tutte (Tutte, 1960, 1963) computed straight line embeddings of planar graphs in which the spring constants were chosen as 1, i.e.,  $D_{ij} = 1$ . Greiner and Hormann (1997) chose the spring constants in proportion to the lengths of the corresponding edges in the triangulated mesh. However, a primary disadvantage of both of these approaches is that they do not satisfy the following minimum requirement expected from any parameterisation method.

**Definition 4.1** *Linear reproduction*

*Suppose that  $S$  is contained in a plane so that its vertices have coordinates  $p_i = (x_i, y_i, 0)$  with respect to an appropriately chosen orthonormal coordinate frame. Then, a globally isometric (thus, optimal) parameterisation can be defined using the local coordinates  $\mathbf{x}_i = (x_i, y_i)$  as the parameter points themselves, that is, by setting  $\mathbf{u}_i = \mathbf{x}_i$  for  $i = 1, \dots, n + b$ . Because the overall parameterisation is a linear function, we say that a parameterisation method has linear reproduction if it produces this isometric mapping in this setting. (Hormann et al., 2008)*

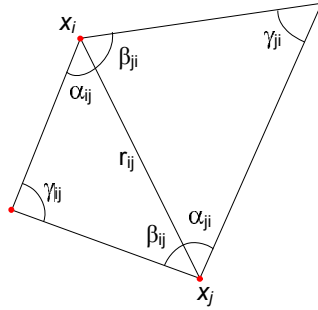
The linear reproduction requirement can be achieved if the parameter points for the boundary vertices are set correctly and the values of  $\lambda_{ij}$  are chosen such



that the following expression is true:

$$\mathbf{x}_i = \sum_{j \in N_i} \lambda_{ij} \mathbf{x}_j \text{ and } \sum_{j \in N_i} \lambda_{ij} = 1 \quad (4.28)$$

for all of the interior vertices. These values,  $\lambda_{ij}$ , are also called the barycentric coordinates of  $\mathbf{x}_i$  with respect to its neighbours  $\mathbf{x}_j, j \in N_i$ . A. F. Möbius introduced (1827) the concept that any position inside a simplex can be uniquely defined as the average point, or barycentre, of the masses placed at its vertices. Therefore, when a vertex  $\mathbf{x}_i$  has exactly three neighbours, its coordinates,  $\lambda_{ij}$ , are uniquely defined. This result is false for any polygon with more than three vertices, i.e., there are certain methods of defining the barycentric coordinates of a point in the polygon. Floater et al. (2006) presented a common framework for the determination of the barycentric coordinates that is briefly shown as follows. Given an interior point  $\mathbf{x}_i$ , let  $\mathbf{x}_j$  be one of its neighbours, let  $r_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$  be the length of the edge  $e_{ij} = [\mathbf{x}_i, \mathbf{x}_j]$ , and let the angles at the corners of the triangles adjacent to  $e_{ij}$  be denoted as shown in Figure 4.1.



**Figure 4.1:** Notation of the angles

The most popular formulas used to define the coordinates include the Wachspress, the discrete harmonic, and the mean value, as shown below.

- Wachspress coordinates:

$$w_{ij} = \frac{\cot \alpha_{ji} + \cot \beta_{ij}}{r_{ij}^2}.$$

- Discrete harmonic coordinates:

$$w_{ij} = \cot \gamma_{ij} + \cot \gamma_{ji}.$$

- Mean value coordinates:

$$w_{ij} = \frac{\tan \frac{\alpha_{ij}}{2} + \tan \frac{\beta_{ji}}{2}}{r_{ij}}.$$

From these values,  $w_{ij}$ , and the barycentric coordinates  $\lambda_{ij}$  of  $\mathbf{x}_i$  are defined by the following formula:

$$\lambda_{ij} = w_{ij} / \sum_{k \in N_i} w_{ik}.$$

The three choices mentioned above can be directly applied to triangulated meshes without projecting each interior vertex and the neighbours of that vertex into the plane. In certain configurations, the Wachspress and discrete harmonic coordinates can receive negative values, whereas the mean value coordinates are always positive. When certain weights,  $w_{ij}$ , are negative, the parameter space can be a self-intersection (invalid parameterisation). However, if all of the weights are positive, and the parameter points of the boundary vertices form a convex polygon, then the parameter space cannot be a self-intersection (valid parameterisation). The latter fact, i.e., a convex boundary, has first been proven in the literature (Tutte, 1963) for the special case of  $\lambda_{ij} = 1/\eta_i$  where  $\eta_i = \#N_i$  is the number of  $p_i$ 's neighbours. Moreover Floater (1997) presented that the proof is also true for arbitrary positive weights  $\lambda_{ij}$ . The Tutte's barycentric mapping theorem is stated as follows:

**Theorem 4.2** *Given a triangulated surface homeomorphic to a disk, if the  $(u, v)$  coordinates at the boundary vertices lie on a convex polygon, and if the coordinates of the internal vertices are a convex combination of their neighbours, then the  $(u, v)$  coordinates form a valid parameterisation (without self-intersections).*

The solvability of the linear systems of Equations 4.26 is an important issue that must be addressed. It has been proven that the matrix  $A$  is guaranteed to be non-singular for discrete harmonic (Pinkall et al., 1993) and mean value coordinates (Floater, 1997).

### Parameterisations based on distortion measures

Another approach to find a parameterisation  $f$  of a surface  $S$  over a parameter domain  $\Omega$  is based on the idea of minimising the overall distortion of the parame-

terisation. Using differential geometry, the distortion is constant per triangle with the singular values  $\sigma_1^t$  and  $\sigma_2^t$ , and the overall distortion is given in the following equation:

$$\overline{E}(f) = \frac{\sum_{t \in \Omega} E(\sigma_1^t, \sigma_2^t) A(t)}{\sum_{t \in \Omega} A(t)}. \quad (4.29)$$

We also consider the overall distortion of the inverse parameterisation  $g = f^{-1}$ ,

$$\overline{E}(g) = \frac{\sum_{T \in S} E(\sigma_1^T, \sigma_2^T) A(T)}{\sum_{T \in S} A(T)}. \quad (4.30)$$

The advantage of the  $\overline{E}(g)$  is that the sum of the surface triangle areas in the denominator is constant; therefore, it can be ignored upon minimisation.

### Harmonic Maps:

The harmonic map parameterisation method considers the Dirichlet energy of the inverse parameterisation  $g$ , which is given by  $\overline{E}(g)$  in Equation 4.30 with the local distortion measure given in the following equation:

$$E_D(\sigma_1, \sigma_2) = \frac{1}{2}(\sigma_1^2 + \sigma_2^2). \quad (4.31)$$

In this case, the energy  $\overline{E}(g)$  is quadratic in the parameter points  $\mathbf{u}_i$ . Therefore, the energy can be minimised by solving a linear system of equations. The inverse  $f = g^{-1}$  of the solution, the harmonic map  $g$ , is exactly the barycentric mapping with discrete harmonic coordinates. The disadvantage of this method is that the boundary of the parameterisation must be fixed in advance. If this requirement is not satisfied, the parameterisation degenerates, because the  $E_D$  value takes its minimum for mappings with  $\sigma_1 = \sigma_2 = 0$  so that an optimal parameterisation is one that maps all of the surface triangles  $T$  to a single point.

### Conformal Maps:

The conformal maps method uses the conformal energy as a local distortion measure in Equation 4.30, as follows:

$$E_C(\sigma_1, \sigma_2) = \frac{1}{2}(\sigma_1 - \sigma_2)^2. \quad (4.32)$$

Minimising this energy leads to solving a linear system of equations. Only two of the boundary vertices must be fixed to have a unique solution. It is clear that the conformal energy  $E_C$  is minimal when  $\sigma_1 = \sigma_2$ ; however, it is not the only

energy that favours conformability. Another energy, called MIPS (Most Isometric ParametrisationS), was introduced by Hormann and Greiner (2000) and uses the following equation:

$$E_M(\sigma_1, \sigma_2) = \frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1} = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1 \sigma_2}. \quad (4.33)$$

This energy is minimal if and only if  $\sigma_1 = \sigma_2$ . An advantage of this distortion measure is the symmetry with respect to inversion, according to the following equation:

$$E_M(\sigma_1^T, \sigma_2^T) = E_M(\sigma_1^t, \sigma_2^t).$$

Thus, this calculation measures the distortion of the mappings  $f_t$  and  $g_T$  simultaneously. The disadvantage of this method is that it results in solving a non-linear problem. However,  $\bar{E}_M(f)$  is a quadratic rational function in the  $\mathbf{u}_i$ , and  $\bar{E}_M(g)$  is a sum of quadratic rational functions. Both of these energies can be minimised with standard gradient descent methods. Moreover, this method guarantees the bijectivity of the resulting map.

### 4.3.2 Cross-parameterisation and compatible remeshing

Constructing a cross-parameterisation between two meshes involves determining the bijective mapping between these meshes. The most common approach is to parameterise both of the meshes on a common base domain, e.g., a simplicial complex or a sphere. The final map is obtained by the composition of these parameterisations. Another approach is based on energy functions by which one mesh is directly attracted towards the other. The energy function consists of components that pull the vertices of one mesh towards the nearest locations on the other while attempting to maximally preserve the shape of the mesh.

Praun et al. (2001) presented a method using a simplicial complex as a common base domain. Given a simplicial complex as a base domain of the two meshes, users draw a network of paths between the feature vertices corresponding to the vertices of the simplicial complex to partition both meshes into triangular patches homeomorphic to disks. A parameterisation function of each patch on the corresponding face of the simplicial complex is constructed using a fixed-boundary parameterisation method. Finally, cross-parameterisation is constructed from these parameterisations.

Kraevoy and Sheffer (2004) extended the method of Praun et al. (2001) to automatically construct the network of paths. Moreover, the method may split the existing mesh edges when required. The input to the methods includes a set of correspondences between the feature vertices on the two input models. The method uses those sets as the vertices of the base complex.

Some methods use an energy-driven approach such as those proposed by Allen et al. (2003), and Sumner and Popović (2004). These methods require the user to specify many point-to-point correspondences between the two input models. These methods work well when the meshes are very similar, and they are sensitive to the weights used inside the energy function, considering the different components.

In the interpolation and the morphing applications, it is insufficient to obtain only a cross-parameterisation between the two meshes. For these applications, the two meshes are required to have the same connectivity. There are three primary approaches to generate this common connectivity, as shown below.

- **Base mesh subdivision:** Praun et al. (2001) used the base mesh connectivity and refined it by the one-to-four subdivision pattern. Therefore, the geometries of both models are captured in as many levels of subdivisions as required. This method is simple, but the disadvantage is that it depends on the shape of the base mesh triangles. To reach to an appropriate precision, the method must generate a large number of triangles, approximately 10 times that of the input mesh sizes.
- **Overlay:** Intersecting the two input meshes in the parameter domain to generate a common connectivity is another approach (Alexa, 2000, Schreiner et al., 2004). The common connectivity is constructed by combining all of the vertices of the two input meshes and the new vertices generated by the intersection. The method exactly preserves the input geometries but is difficult to implement, and the number of triangles is excessively large.
- **Remeshing:** Kraevoy and Sheffer (2004) proposed an alternative method in which the connectivity of one input meshes is used as a basis and then refined as required based on an approximate error with respect to the second mesh. The resulting mesh depends on the input mesh chosen as the basis, and it is only an approximation of the target mesh.

## 4.4 TGSIS morphological interpolation

This interpolation method was proposed to build up 4-dimensional model from 3-dimensional sections given at time instances. The 4-dimensional models are the input data for TGSIS. The method directly uses triangulated surfaces which represent geological objects. It is also tailored for geological interpreted surfaces with geometrical constraints. In this section, the underlying concepts of the method are introduced.

Let  $M$  be a triangulated mesh in  $R^3$ . Methods to represent a mesh have been developed, with examples described in (Botsch et al., 2010). In short, we consider a mesh to be a pair of a sequence of its vertices and its topology,  $M = (V, T)$ .  $V$  is a sequence of  $n$  distinct points,  $v_i = (x_i, y_i, z_i)$  in  $R^3$ .  $V$  can also be considered a map from the index set  $I = \{1 \dots n\}$  to  $R^3$ , where  $V(i) = v_i$  for all  $i = 1, \dots, n$ .  $T$  is the topology or the structure of the mesh entirely defined on the index set  $I$ . The topology defines the set of triangles or faces, the set of edges, the set of boundaries and the set of interior/boundary vertices of the mesh (see Section 2.5.4).

The method models data in the time interval  $[t_i, t_{i+1}]$  from data at time instances  $t_i$  and  $t_{i+1}$  by smoothly changing a source mesh into a target mesh. Choosing data at  $t_i$  as the source mesh and data at  $t_{i+1}$  as the target mesh, or vice versa, depends on the user. However, the algorithm is designed in a way that it will become faster (because cutting paths are not required in the target mesh) when the more complicated mesh (more vertices or complicated topology) is chosen as the source mesh. We always denote the source mesh as  $M_s$  and the target mesh as  $M_t$ . The following paragraphs define some of the special terms used to describe the algorithm.

A *control vertex* of the mesh  $M(V, T)$  is a user-selected vertex  $v_i \in V$ . A *path* in mesh  $M(V, T)$  is a sequence  $(v_{p_1}, v_{p_2}, \dots, v_{p_h})$ , where  $p_i \in I$  and  $[v_{p_i}, v_{p_{i+1}}]$  is an edge of  $M$  for all  $i = 1, \dots, h - 1$ . A *Boundary path* is a path in which all of its vertices are in the boundaries. The *cutting path* and *fence path* are paths used in the cutting procedure and the partition procedure, respectively.

A *unit regular  $k$ -polygon*,  $k \geq 3$ , is a simple polygon with  $k$  vertices in a certain

plane, such that coordinates of its vertices are defined by the formula,

$$v_i = (\cos(\frac{2\pi}{k}(i-1)), \sin(\frac{2\pi}{k}(i-1))),$$

for all  $i$  in  $\{1 \dots k\}$ .

The *error*  $\varepsilon(M, N)$  between two meshes,  $M$  and  $N$ , is defined by the form

$$\varepsilon(M, N) = \max\{\max\{\text{dist}(x, N) : x \in V_{M_I}\}, \max\{\text{dist}(x, M) : x \in V_{N_I}\}\},$$

where  $V_{M_I}$  is a set of interior vertices of  $M$ ,  $V_{N_I}$  is a set of interior vertices of  $N$  and  $\text{dist}(x, L)$  (the distance between a point  $p$  and a mesh  $L$ ) is the minimum distance between the point  $p$  and those in  $L$ . Note that in this term, we consider only interior vertices, not boundary vertices.

**Definition 4.2** Two meshes,  $M(V, T)$  and  $N(U, G)$ , are compatible, if (i)  $V$  and  $U$  have the same total number of vertices and they are corresponding in their order, i.e., a trivial map,  $h$ , exists, such that  $h(v_i) = u_i$  for all  $i = 1, \dots, n$ ; and (ii)  $T = G$ .

**Definition 4.3** Given  $m \geq 2$  and an  $n$ -vertices mesh  $M(V, T)$ , trajectories of the mesh  $M(V, T)$  that represent its continuous evolution into its compatible mesh  $N(U, T)$  are  $n$  distinct line strings, such that for each line string,  $m$  vertices exist, i.e., the  $i^{\text{th}}$  line string,  $p^i = (w_1^i, w_2^i, \dots, w_m^i)$ , and  $w_1^i = v_i, w_m^i = u_i$  for all  $i = 1, \dots, n$ .

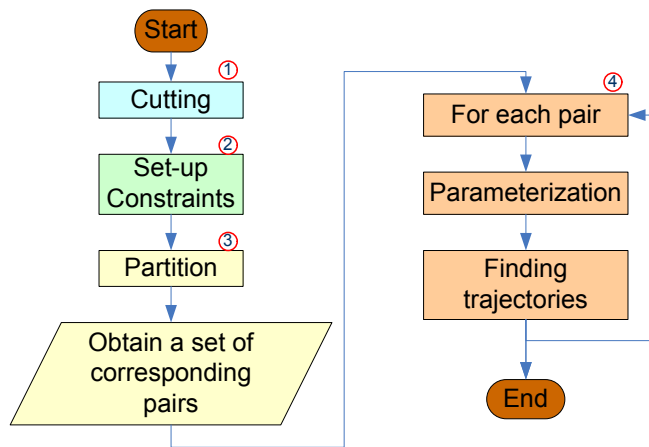
**Definition 4.4** Given  $m \geq 2$ , an  $n$ -vertices mesh  $M(V, T)$  and its compatible mesh  $N(U, T)$ , linear trajectories of the mesh  $M(V, T)$  are trajectories of  $M$ , where each vertex of a trajectory,  $p^i = (w_1^i, w_2^i, \dots, w_m^i)$ , is defined as

$$w_j^i = v_i + \frac{j-1}{m-1}(u_i - v_i)$$

for all  $i = 1, \dots, n; j = 1, \dots, m$ .

Figure 4.2 presents a block diagram of the algorithm containing the four main procedures. First, the cutting procedure adds “missing” boundaries to the meshes, such that the error between the old and new meshes is zero. Second, the setting up constraints procedure sets up control vertices, the pairs of control vertices between

the source mesh and the target mesh, fence paths and the attaching constraints of boundary vertices. Third, the partition procedure subdivides the source mesh and the target mesh into corresponding pairs of source patches and target patches, such that every patch is homeomorphic to a disk. Fourth, the calculating procedure calculates cross-parameterisation, generates compatible meshes and finds trajectories for each pair of patches. Finally, these results are combined to achieve the final result for the original source mesh and the original target mesh. The first two procedures are user interactive procedures, while the last two are automatic. Moreover, the first two create input data for the last two.



**Figure 4.2:** The block diagram of the algorithm

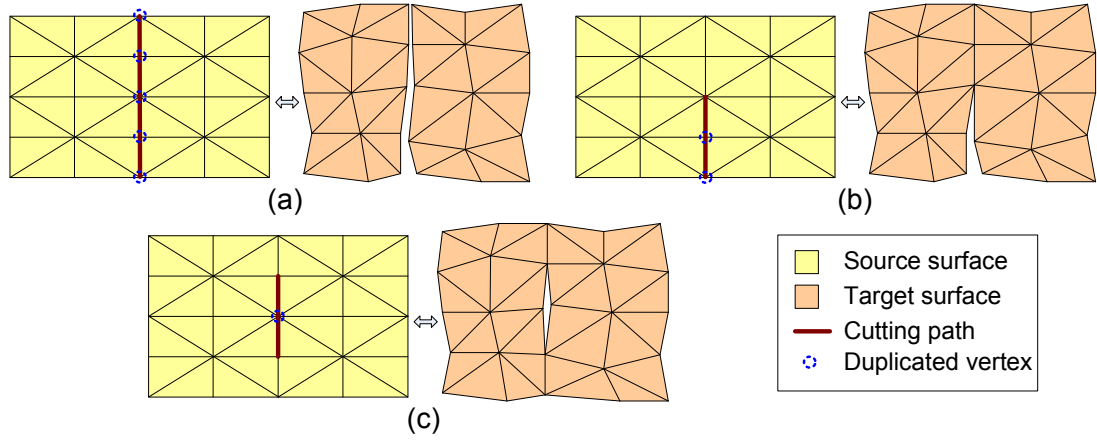
### 4.4.1 Cutting

Missing boundaries of the source mesh and the target mesh exist, due to the faulting or fracturing phenomena. Cutting paths are created to represent these missing boundaries. They are created in the source mesh to represent naturally occurring faults or fractures that first begin appearing in the source surface and then evolve into fractures in the target surface. Cutting paths are not required to be created in the target mesh to represent faults or fractures already existing in the source surface and evolving to disappear in the target surface. In the latter situation where cutting paths are not required, fence paths are created instead.

Let  $P$  be a set of user-selected cutting paths in  $M$ . This procedure cuts the mesh  $M$  by  $P$ . Three types of cutting paths are supported, as shown in Figure 4.3. All vertices and edges of cutting paths are duplicated (as in Figure 4.3a), except for a one-end vertex (as in Figure 4.3b) or two-end vertices (as in Figure 4.3c).



Blue dashed circles represent duplicated vertices.



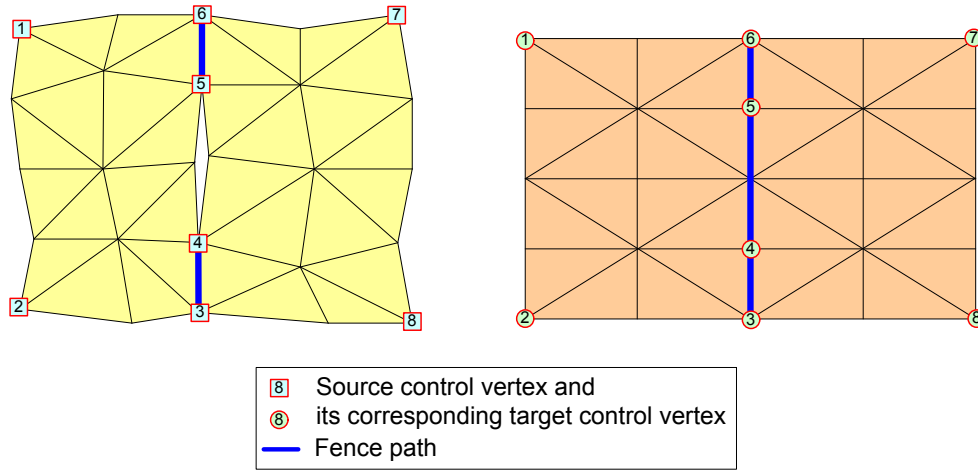
**Figure 4.3:** Cutting meshes by cutting paths. (a) All vertices and edges are duplicated. (b) All vertices and edges, except for a one-end vertex, are duplicated. (c) All vertices and edges, except for two-end vertices, are duplicated

Because the procedure is completed as described above, the error between the old mesh and the new mesh is zero. After this procedure, we still denote  $M_s$  and  $M_t$  as the source mesh and the target mesh, respectively.

#### 4.4.2 Setting up Constraints

The algorithm uses four types of constraints: *control vertex*, *control vertex pair*, *fence path*, and *attaching constraint*. The user selects all of the constraints manually with automatic tools, such as finding the shortest path between two-end vertices. Control vertices and control vertex pairs present fixed (known) corresponding vertices and are used to control the correspondence of other vertices. Control vertices are often corner vertices. The control vertices in the source mesh and the target mesh are represented by two sequences of control vertices,  $V_c$  and  $W_c$ , respectively, such that they deduce control vertex pairs by their order, i.e.,  $(v_{c,i}, w_{c,i})$  is a control vertex pair. Fence paths are used to alter cutting paths in the target mesh or to partition the source mesh and the target mesh into patches that are homeomorphic to disks. Fence paths are required to connect exactly two control vertices, for example, a fence path with  $h$  vertices in the source mesh,  $M_s$ ,  $p(v_{p_1}, v_{p_2}, \dots, v_{p_h})$ , satisfies the following conditions:  $v_{p_1}, v_{p_h} \in V_c$  and  $v_{p_i} \notin V_c$  for all  $i = 2, \dots, h - 1$ . All fence paths of the source mesh or target mesh cut each other only at their end vertices. Figure 4.4 depicts an example of control

vertices, control vertex pairs (by corresponding order) and fence paths.

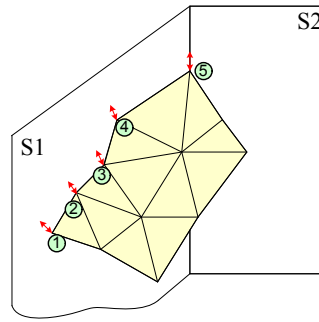


**Figure 4.4:** An example of control vertices, control vertex pairs and fence paths

The attaching constraints present the adhesion of the vertices of the source mesh to the controlling surfaces during the evolution of the source mesh. If a 1-controlling surface attaching constraint is imposed on a vertex of the source mesh, this vertex will always be located on the controlling surface during its evolution. Similarly, if a 2-controlling surface attaching constraint is imposed on a vertex of the source mesh, this vertex will always be located on the line that is the intersection of the two controlling surfaces. The method delimits that the attaching constraints are only imposed on boundary vertices of the source mesh and that there are a maximum of two controlling surfaces involved in each attaching constraint. Furthermore, all vertices attached to two surfaces must be control vertices. Through these attaching constraints, each boundary vertex of the source mesh attaches to no surface, one surface or two surfaces. Figure 4.5 presents an example of attaching constraints. In this figure, boundary vertices, 1, 2, 3 and 4, attach to one surface ( $S_1$ ), (i.e., 1-controlling surface attaching constraint), boundary vertex 5 attaches to two surfaces ( $S_1, S_2$ ), (i.e., 2-controlling surface attaching constraint) and the other boundary vertices do not attach to any surface, meaning that there are no attaching constraints imposed on these boundary vertices.

### 4.4.3 Partition

This procedure starts with the DefinePatches sub-procedure, dividing both the source mesh and the target mesh into patches by fence paths and their boundaries.



**Figure 4.5:** An example of attaching constraints

---

```

Procedure DefinePatches(Mesh M, Set_of_Fence F, Set_of_Mesh P)
{
    //input a mesh  $M(V,T)$ ; a set of fence paths  $F$ 
    //output a set of meshes or a set of patches  $P$ 
    Attach an integer field named patchid to all edges of M.
    For each edge e in M{
        if e in any fence f of F then e.patchid = -1;
        else e.patchid = 0;
    }
    PATCH = 0; STACK = Empty;
    For each edge e in M{
        if (e.patchid == 0) then{
            PATCH=PATCH + 1;
            e.patchid = PATCH; STACK.push(e);
        }
        While (not STACK.isEmpty()){
            e1 = STACK.pop();
            For each edge e2 in each triangle containing e1
                and e2.patchid = 0
            {
                e2.patchid = PATCH;
                STACK.push(e2);
            }
        }
    }
    Create a set of mesh P from the values in field patchid
        of all edges of M.

```

---

}

---

After the DefinePatches sub-procedure, the source mesh and the target mesh are partitioned into two sets of patches. This sub-procedure is consistently successful for any type of mesh and set of fence paths. Next, we check if each patch contains at least three control vertices and that all control vertices are in its boundary. We also check if each patch is homeomorphic to a disk. If the check fails, the first two procedures, i.e., cutting and set-up constraints, need to be repeated. Some reasons for this failure are as follows: (1) there are too few control vertices; (2) there exists a control vertex, which is not on the boundaries or fence paths; or (3) a hole exists in a patch. The requirement of disk homeomorphism can be satisfied for any type of mesh through the cutting procedure.

After all of the above steps, meshes  $M_s$  and  $M_t$  are partitioned into the same number of patches,  $k$ . If this condition is not satisfied, all of the above steps need to be repeated to set up more fence paths. This condition can be satisfied for any type of mesh through the set of user-defined fence paths. Discovery of the corresponding patches is accomplished by identifying the control vertices and their order in each patch. The result of this step is a set of patch pairs  $(P_{i,s}, P_{i,t})$  for all  $i$  in  $\{1 \dots k\}$ . Once again, if such a set of patch pairs is not found, all of the above steps are repeated. The cause of this failure is the lack of correspondence between the control vertices. This step will be successful if the control vertices and their order are correct.

#### 4.4.4 Calculating

In this procedure, we work with each source patch and its corresponding target patch pair, denoted by  $(L_s, L_t)$ . This procedure contains two sub-procedures: calculating the cross-parameterisation (or constructing compatible meshes) sub-procedure and the finding trajectories sub-procedure.

##### Calculating Cross-Parameterisation or Constructing Compatible Meshes

This sub-procedure is used to construct a new mesh that is compatible with the source mesh and that approximates the target mesh. The new mesh and the

source mesh are compatible in the sense of having one-to-one correspondence between their vertices, edges and faces (Definition 4.2). The new mesh and the target mesh are considered well approximated if the distance or the error between them is smaller than the user's threshold. Splitting a few triangles of the source mesh will reduce the error between the new mesh and the target mesh. In the following paragraphs, more detailed descriptions of the algorithm are given.

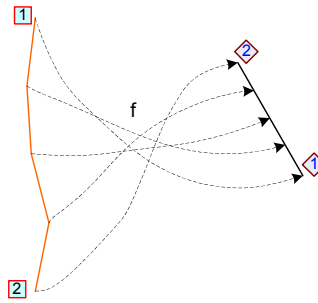
Given two patches or triangle meshes,  $L_s(V, T), L_t(W, F)$ , both are homeomorphic to disks. Let  $I_{cs} = (i_{cs,1} \dots i_{cs,k})$  and  $I_{ct} = (i_{ct,1} \dots i_{ct,k})$  be index sequences of  $k, k \geq 3$ , with distinct control vertices in  $L_s$  and  $L_t$ , respectively; additionally, a correspondence exists between the index sequences by their order. All control vertices are in the boundary of their meshes and are in a clockwise or counterclockwise order. To construct compatible meshes, we first find a bijection between the two meshes and, then, construct a new mesh using this map.

In the first step, we map both patch boundaries to the boundary of a unit regular  $k$ -polygon using the “chord” length method for each boundary path (Figure 4.6), i.e., mapping  $f$  from a path  $p(v_{p_1}, v_{p_2}, \dots, v_{p_h})$  with  $h$  vertices to a segment  $s(x_1, x_2)$  of the unit regular  $k$ -polygon. This method is defined by the following formula:

$$f(v_{p_1}) = x_1; f(v_{p_i}) = f(v_{p_{i-1}}) + \frac{\text{length}(v_{p_i}, v_{p_{i-1}})}{\text{length}(p)}(x_2 - x_1)$$

for all  $i$  in  $2, \dots, h$ , where

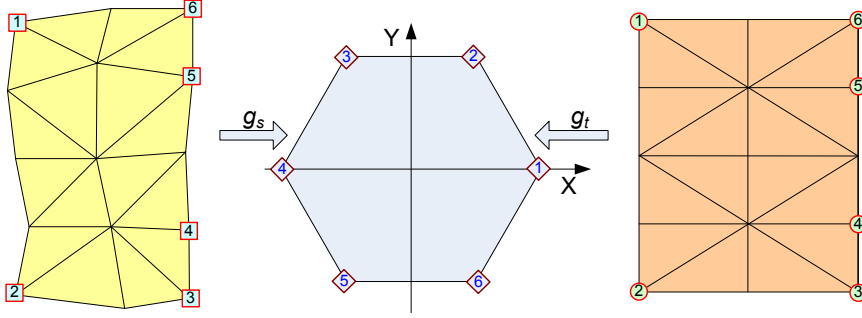
$$\text{length}(p) = \sum_{j=2}^h \text{length}(v_{p_j}, v_{p_{j-1}}).$$



**Figure 4.6:** An example of the “chord” length method

Subsequently, we use mean value parameterisation (Floater, 1998, 2003) to

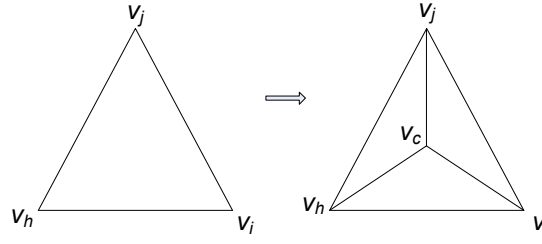
obtain bijective maps,  $g_s$  and  $g_t$ , from  $L_s$  and  $L_t$ , respectively, to this unit regular  $k$ -polygon. The composition map,  $g = g_t^{-1} \times g_s$ , is a bijection from  $L_s$  to  $L_t$ . Figure 4.7 displays an example of these parameterisations.



**Figure 4.7:** Parameterisations to the unit regular  $k$ -polygon

In the second step, we construct a new mesh  $L_{st}$  by a set of its vertices, which is an image of  $V$  through  $g$ ,  $U = g(V)$ , and the topology  $T$  of  $L_s$ , i.e.,  $L_{st}(U, T)$ . Note that  $L_{st}$  is a compatible mesh of  $L_s$ ; all of its vertices are on  $L_t$ , and it is an approximate mesh of  $L_t$ . We require that  $L_{st}$  is a “good” approximation of  $L_t$ , i.e., the error between  $L_{st}$  and  $L_t$  is smaller than the user’s threshold. Because all of the vertices of  $L_{st}$  are on  $L_t$ , the distance from every vertex of  $L_{st}$  to  $L_t$  is zero. Therefore, the error between  $L_{st}$  and  $L_t$  is the maximum of the distances from the interior vertices of  $L_t$  to  $L_{st}$ , as shown in the definition of terms. To obtain a smaller error than the user’s threshold, we complete the following procedures. For each interior vertex  $w_i$  of the target mesh  $L_t(W, F)$ , if the distance  $d_i$  from  $w_i$  to the mesh  $L_{st}(U, T)$  is greater than the user’s threshold  $\varepsilon$  and the triangle  $(v_i, v_j, v_h)$  contains a point  $p$  where  $\|w_i, p\| = d_i$ , we subdivide a triangle  $(v_i, v_j, v_h)$  into three triangles by inserting a new point at its centroid and updating  $L_s$  (clearly, the error between the old  $L_s$  and the updated  $L_s$  is zero, so for clarity, we still denote the updated mesh by  $L_s(V, T)$ ). Figure 4.8 presents an example of this subdivision. If such a modification has been completed, then the new mesh  $L_{st}$  would need to be constructed and checked for error again. Notice that this subdivision reduces the error between  $L_t$  and  $L_{st}$  and does not affect other patches.

After the above two steps,  $L_s(V, T)$  and  $L_{st}(U, T)$  are compatible meshes, and the error between  $L_{st}$  and  $L_t$  is smaller than the threshold  $\varepsilon$ . This mesh pair can be represented by the mesh  $L_s(V, T)$  and a sequence of displacement vectors, where each displacement vector is the difference of a vertex  $u$  in  $U$  and its corresponding



**Figure 4.8:** Splitting a triangle  $(v_i, v_j, v_h)$  into three new triangles,  $(v_i, v_j, v_c)$ ,  $(v_j, v_h, v_c)$ ,  $(v_h, v_i, v_c)$ , by inserting a new point,  $v_c$ , at the centroid of triangle  $(v_i, v_j, v_h)$

vertex  $v$  in  $V$ .

### Finding Trajectories

This sub-procedure is used to find paths in which vertices of the source mesh evolve into their corresponding target vertices. In the case of no attaching constraints, i.e., the source mesh is free to evolve from its original shape into its target shape, trajectories of all its vertices are defined by their linear trajectories, and the rest of this calculation can be ignored. When some attaching constraints are used to constrain the geometry of the source mesh during its evolution, trajectories of vertices need to be calculated from their linear trajectories. Trajectories of vertices, which impose 1- or 2-controlling surface attaching constraints, are projected lines of their linear trajectories to the controlling surface or the intersection line of the two controlling surfaces, respectively. Trajectories of boundary vertices without attaching constraints are their linear trajectories. Trajectories of all interior vertices are calculated by solving sparse linear systems of equations. The algorithm is described as follows.

Given a triangle mesh  $L_s(V, T)$ , where  $V = (v_1, v_2, \dots, v_n)$  is a sequence of  $n$  vertices, let  $m$ , the number of vertices of each trajectories, be a user-defined integer,  $m \geq 2$ . By using attaching constraints, the vertices set  $V$  of  $L_s$  can be subdivided into disjoint subsets:

$$V = V_I \cup V_{B_0} \cup V_{B_1} \cup V_{B_2}$$

where  $V_I$  is the set of interior vertices,  $V_{B_0}$  is the set of boundary vertices without attaching constraints,  $V_{B_1}$  is the set of boundary vertices with 1-controlling surface attaching constraints and  $V_{B_2}$  is the set of boundary vertices with 2-controlling surface attaching constraints.

First, trajectories of vertices in  $V_{B_0}$ ,  $V_{B_1}$  and  $V_{B_2}$  are initialized by connecting the vertices of  $V_{B_0}$ ,  $V_{B_1}$  and  $V_{B_2}$  to their corresponding vertices to create line segments. Then, we subdivide these line segments into  $m - 1$  equal length sub-segments, i.e., linear trajectories are constructed for each boundary. Subsequently, trajectories of vertices in  $V_{B_1}$  are modified by replacing their vertices with their projection into controlling surfaces; the trajectory of each vertex in  $V_{B_2}$  is modified by changing its vertices to its projection into the curve, which is the intersection of the two controlling surfaces of the vertex.

We calculate trajectories of vertices in  $V_I$  by calculating  $m - 2$  intermediate meshes when the mesh  $L_s(V, T)$  evolves into its corresponding mesh,  $L_{st}(U, T)$ . We label  $L_s$  with  $L^1$ ,  $L_{st}$  with  $L^m$  and intermediate meshes with  $L^t$  for each  $t = 2, \dots, m - 1$ . The boundary of the mesh,  $L^t$ , is defined by trajectories of vertices in  $V_{B_0}$ ,  $V_{B_1}$  and  $V_{B_2}$ . We also assume that each interior vertex of  $L^t$  is a convex combination of its neighbors as follows. Let  $N(i)$  be the set of vertex indices of the neighborhood of vertex  $v_i^t$ , and let  $I_I$  be the index set of  $V_I$ . A set of non-negative real values,  $\lambda_{ij}^t$ , exists, such that:

$$\sum_{j \in N(i)} \lambda_{ij}^t = 1$$

and Equation 4.34 below is satisfied for all  $i$  in  $I_I$ :

$$\sum_{j \in N(i)} \lambda_{ij}^t v_j^t = v_i^t. \quad (4.34)$$

We calculate  $\lambda_{ij}^t$  based on the values,  $\lambda_{ij}^1$  and  $\lambda_{ij}^m$ , from the first mesh and the last mesh,  $L^1$ ,  $L^m$ , as in Equation 4.35:

$$\lambda_{ij}^t = \lambda_{ij}^1 + \frac{t - 1}{m - 1} (\lambda_{ij}^m - \lambda_{ij}^1) \quad (4.35)$$

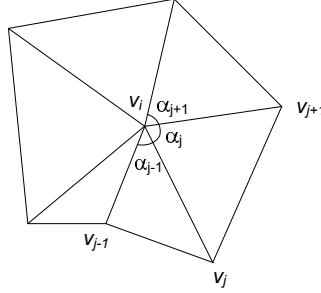
Values  $\lambda_{ij}^1$  and  $\lambda_{ij}^m$  are calculated from  $L^1$  and  $L^m$  using the mean value as described in (Floater, 2003) with Equation 4.36 and notations in Figure 4.9.

$$\lambda_{ij} = \frac{w_j}{\sum_{k \in N(i)} w_k}, \quad w_j = \frac{\tan(\alpha_{j-1}/2) + \tan(\alpha_j/2)}{\|\nu_j - \nu_i\|} \quad (4.36)$$

Because  $0 \leq \alpha_{j-1}, \alpha_j \leq \pi$ ,  $\lambda_{ij}$  in Equation 4.36 is defined and non-negative,  $\lambda_{ij}^t$  in Equation 4.35 is non-negative for each  $t = 2, \dots, m - 1$ . Equation 4.34



gives a sparse linear system of equations, which can be solved sufficiently by a solver, such as OpenNL (OpenNL, 2014) or Eigen (Guennebaud et al., 2010). This system of equations has a unique solution, proved in (Floater, 1997, 1998). Because of the unique solution, all trajectories of  $L$  have been defined.



**Figure 4.9:** Angle notations

Note that by using mesh representations where the topology is explicitly stored, determining the neighborhood,  $N(i)$ , of vertex  $v_i$  is trivial. Such mesh representations include G-Maps, C-Maps, Cell-Tuple-Structure, Halfedge data structure.

## 4.5 Summary

In this chapter, we presented two primary approaches for morphological interpolation. One approach is based on the mathematical morphology theory, which operates on multi-dimensional binary images (sets). This approach can be applied to triangulated meshes through a converting procedure. Another approach is based on the mesh processing techniques, which directly manipulate triangulated meshes. Typically, morphological interpolation methods in this approach include the following two steps: (1) constructing the compatible meshes, and (2) finding the trajectories.

This chapter also reviewed our morphological interpolation method, called TGSIS morphological interpolation that was adapted to represent geological objects existing in the 3-dimensional Euclidean space and evolving in the 1-dimensional time axis. The method can be used in TGSIS to roughly model spatio-temporal data with constraints in terms of geometry (without physical and mechanical constraints). This method is useful when no physical or mechanical process models are available or when there are insufficient data for these models. The method consists of four primary procedures in which the first two procedures are semi-

manual and the last two procedures are automatic. The highlights of the method include a new cutting procedure and a new sub-procedure for finding trajectories using convex combinations. Due to the cutting procedure, the method can work with arbitrary meshes. By partitioning meshes into patches, i.e., triangle meshes homeomorphic to disks, the method reduces the original problem to smaller and simpler problems. The primary calculations include computing the parameterisation sub-procedure and then finding the trajectories sub-procedure; the complexities of both of these calculations are equal to the complexity of a convex, fixed-boundary parameterisation method, e.g., the mean value parameterisation method. The timings of certain parameterisation algorithms are presented in the literature (Hormann et al., 2008).

The TGSIS morphological interpolation has an advantage is that it manipulates triangulated mesh directly and allows end-users/applications to control the interpolation through constraints. Currently, this method is the only method implemented in our prototype software to model temporal data. The modeled data are then stored inside the PostgreSQL system and can be further manipulated using the gOcad system.

# Chapter 5:

## Long Transaction

---

### 5.1 Introduction

#### 5.1.1 Short and long transactions

A *transaction* is a key concept of nearly all database management systems (DBMS). A transaction is a group of queries and modification actions, i.e., SELECT, INSERT, UPDATE, and DELETE actions. A transaction can be characterised in terms of the so-called “ACID” properties. The first of these properties, *Atomicity*, indicates the all-or-nothing execution of transactions. The second property, *Consistency*, indicates that the transactions transform the database from one consistent state to another. The third property, *Isolation*, indicates that each transaction must appear to be executed as if no other transaction is being executed simultaneously. The fourth property, *Durability*, indicates that once a transaction successfully commits, the DBMS guarantees that its results will never be lost, regardless of system failures (Garcia-Molina et al., 2009, PostgreSQL, 2014). A module of a DBMS, called the logging module, supports the first and the fourth properties of a transaction. A module called *concurrency control* supports the second and third properties. By compromising the performance of the system, i.e., the throughput and response time, nearly all database management systems provide certain levels of isolations and the corresponding consistencies/inconsistencies (Adya et al., 2000, Berenson et al., 1995). The decision of which isolation level is appropriate for an application depends on the designers of that applica-

tion.

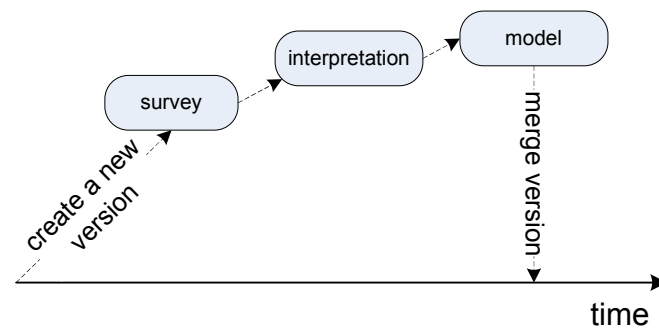
Because nearly all database management systems use either lock-based or multi-version methods for the concurrency control module, a transaction duration is as short as possible, and a transaction is limited to one working session only (Cahill, 2009, Cahill et al., 2008, Ports and Grittner, 2012). Therefore, transactions in the current database management systems are understood as *short transactions* in this chapter. A short transaction begins by performing the BEGIN TRANSACTION command implicitly or explicitly and ends by performing either the COMMIT or ROLLBACK command.

In contrast to a short transaction, *long transactions* typically span a number of days and several working sessions. Long transaction can be considered a group of changes that accumulate over a long working time and are isolated from other changes. Long transactions are implemented based on the concept of database versions that are a logical copy of the database with a group of changes. The CREATEVERSION command starts a new long transaction. Then, there are certain short transactions that may have occurred in several working sessions on different days. The long transaction is finished by either the MERGEVERSION or DELETEVERSION command. In a long transaction, all of the results of the intermediate actions are persistently stored until the end of the transaction (ESRI, 2014c, Oracle, 2013a,b).

### 5.1.2 The benefit of long transactions for TGSIS

#### Isolating a group of changes

Nearly all data collection/modification processes in TGSIS are in the appropriate workflows. Each workflow involves collections of actions, some executed by computer alone, some involving human interaction, and perhaps some being human action alone. The data manipulated by these actions should be shared among the collaborative working group but should not be published or shared with other groups until the process is completed. For example, a geological department starts a project to improve the precision of the structural model in an area. Long transaction supports the project by first creating a new version, then managing the intermediate changes and data (seismic profiles and well markers) and finally updating the production data by the result of the project (Figure 5.1).

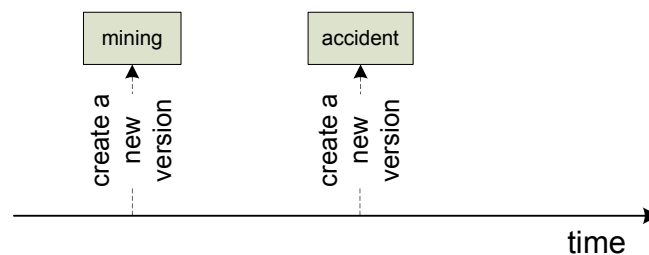


**Figure 5.1:** A data modification process supported by a long transaction

### Storing important historical states

In the geosciences, many important decisions, such as mine production and the assessment of environmental sustainability, are made depending on the state of the current database. Occasionally, an incorrect decision is made, and the mining operation causes unexpected destruction or pollution. If an insurance event occurs, an investigation into whether the complication should have been predicted by the knowledge and data available at the time of the decision must occur. Therefore, storing important state of the database information must be implemented for these types of applications.

By creating new versions at specific times, the long transaction creates a logical copy of the database at those times for the archiving storage purpose. Figure 5.2 shows an example of the storage of two database states, one at the beginning of a mining event and the other at the time an accident occurred.

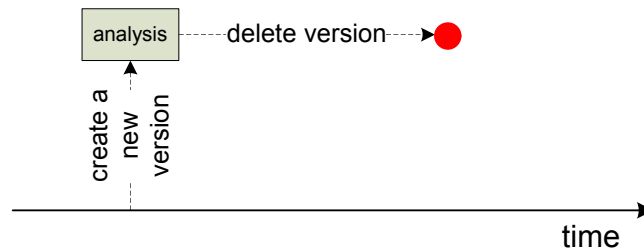


**Figure 5.2:** Long transaction to store historical states

### Creating multiple scenarios

In certain geoscience analyses, simulation scenarios along with data updating actions should be established. The long transaction can support this ability by

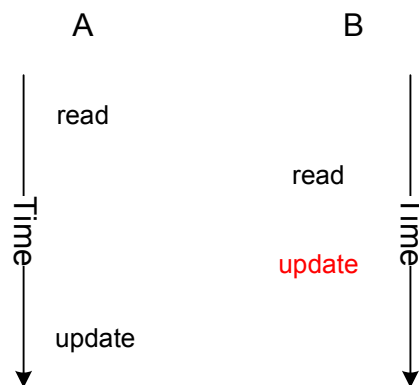
creating a new version, performing the analysis in this version and, at the end, removing this version. A diagram of the “what-if” analysis is shown in Figure 5.3. By creating new versions, the long transaction also provides support for multiple application testers to use the same data set.



**Figure 5.3:** A diagram of the “what-if” analysis

### Preventing the “lost-update” phenomenon

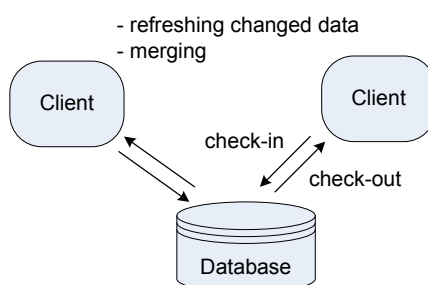
The “lost-update” phenomena occur when an application uses obsolete data to update the data themselves. The result is that the current data appear to have never been observed or the data are lost. For example, user A using gOcad (Paradigm-GOCAD, 2014) retrieves a geological surface. Subsequently, it is assumed that user B, also using gOcad, retrieves the same surface. However, user B’s update is written before user A’s update is written. Therefore, the changes made by user B are silently overwritten by the changes performed by user A (Figure 5.4).



**Figure 5.4:** Lost-update phenomena

Many geoscience applications modify data using a check-out-check-in workflow. In this workflow, large amounts of data must be transferred from a database to a client computer (check-out) for manipulation for a given time and then

uploaded back to the database (check-in). Even when the check-out-check-in workflow occurs in a single working session, this workflow has enormous potential for the lost-update phenomenon to occur. The long transaction can create a template version for each check-out action. When the check-in action occurs, the database will merge the template version into the main database and remove that template version. This approach prevents the lost-update phenomenon by notifying the check-in application whether its data have been changed since its check-out. The check-out-check-in diagram is shown in Figure 5.5.



**Figure 5.5:** Check-out-check-in workflow with refreshing and merging actions

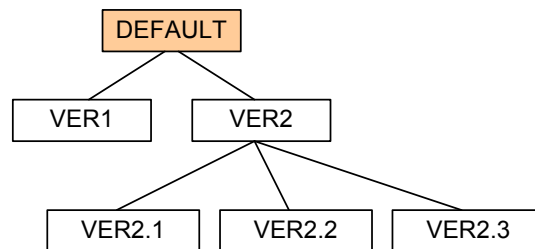
## 5.2 Terminology

As shown in the previous section, a long transaction is a group of short transactions (changes) and is isolated from other long transactions. Therefore, each long transaction appears to be working in a logical database, and it can view all of its changes but cannot view changes made by other long transactions. Oracle Workspace Manager (OWM) calls that working environment a *workspace* (Oracle, 2013a,b), and ArcGIS/ArcSDE (ESRI, 2014c) and TGSIS call it a *version*. To refer to each change in that environment, OWM uses the term *version* for each database state, whereas ArcGIS/ArcSDE uses the term *state*, and TGSIS uses the term *revision*. In the remainder of this chapter, we use the terms used by TGSIS, but when presenting a concept related to OWM and ArcGIS/ArcSDE, their original terms are used for easy reference. The term “transaction” denotes the short transaction.

A *database version* is a logical copy of the database without data duplication. Each version provides all of the transaction properties to its applications, e.g., the consistency property in each database version. All of the versions of a database are maintained under a DBMS as a normal database and have the same data schema

but their data may be different. For example, a database contains three tables, SEISMIC, DRILLING, and SURFACE. The SEISMIC table contains the source seismic profiles surveyed in an interested area. The DRILLING table contains the drilling markers of the area. The SURFACE table contains the geological horizons interpreted from the seismic and drilling data. A horizon, which has a specific age, can be interpreted as a folded surface or a faulted surface, depending on the authors, the geological paradigms, and the available data at the interpreted time. All of the different interpreted horizons and their corresponding source data must be maintained in the database and must be consistent in each group or database version.

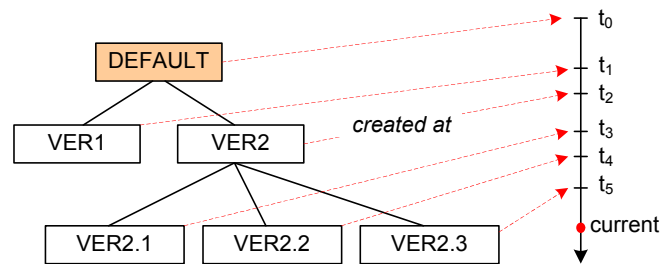
Versions of a database are named by the applications, and they are created and maintained over the life-time of the database. Each version is distinguished from other versions by a set of changes made to that version since it was created. Therefore, versions are always organised as a tree. The root of the version tree is the pre-defined version, named DEFAULT or LIVE, and is created simultaneously with the creation of the database. This type of version tree is shown in Figure 5.6. All of the changes that can be observed by a version are the union of the changes described as follows: (1) the changes made to that version itself; (2) all of the changes made to the parent of the version over the duration, from the parent being created to the time when that version was created; (3) all of the changes made to the grandparent over the duration from the grandparent being created to the time that the parent was created; and so on. For example, version VER2.1 in Figure 5.7, at the current time, can view the changes made to VER2.1, changes made to VER2 over the duration  $[t_2 - t_3]$ , and changes made to DEFAULT over the duration  $[t_0 - t_2]$ .



**Figure 5.6:** Versions are organised as a tree

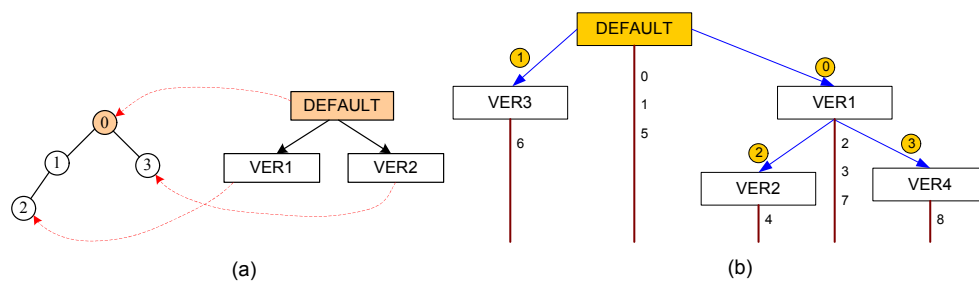
Changes made to a version must be grouped into certain sets to define which changes can be viewed by a child version. These sets are distinguished by the term *revision*, which can be understood as a fine-grained level of the version.





**Figure 5.7:** The version tree and the creation times of the versions

Revisions are determined by the number and are organised as a tree that is either independent from the version tree or based on the version tree. ArcGIS/ArcSDE uses the independent revision tree and maintains the pointers from the version tree to the revision tree. In Oracle Workspace Manager and TGSIS database versioning, the revision tree is based on the version tree and there are implicit pointers from the version tree to the revision tree (Figure 5.8).

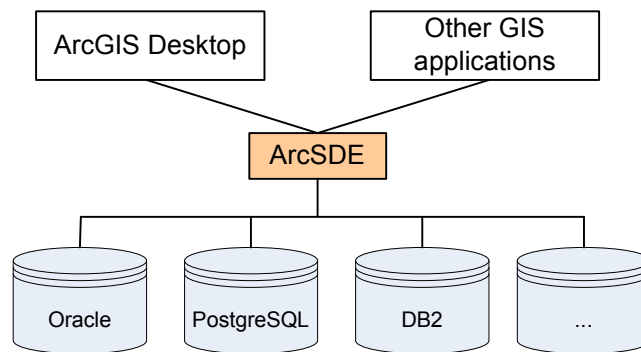


**Figure 5.8:** Version tree and revision tree. (a) in ArcGIS/ArcSDE; (b) in OWM and TGSIS

A versioned database can contain the normal tables and the versioned tables. All of the versions of the database view the normal tables identically. Versioned tables are visible from all of the versions of the database with the same data schema. The rows of a versioned table that are viewed from different versions are different. The direct method used to group rows together into versions is the addition of extra columns to the versioned table and the use of the information in these columns to determine the version to which a row belongs. This method is used in the OWM and TGSIS database versioning. Another method is to leave the versioned table in its original form and store any changes to that table in separate tables, called the *Adds* and the *Deletes* tables. The *Adds* table stores all of the additions and modifications that are made by the insert and update actions to the versioned table. The *Deletes* table stores all of the references to deleted and modified rows. ArcGIS/ArcSDE uses this method for database versioning.

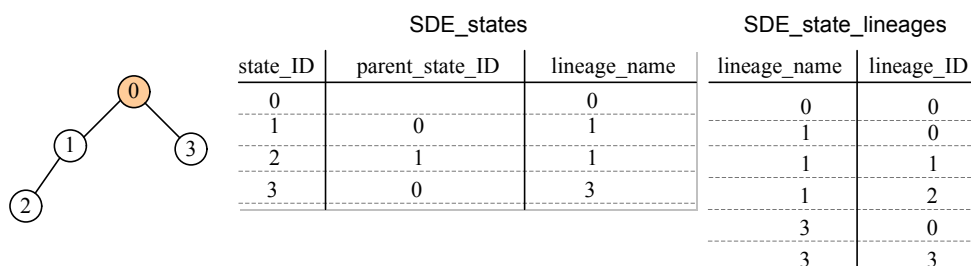
### 5.3 ArcGIS/ArcSDE versioning

The ArcGIS product family (ESRI, 2014b) is one of the most widespread commercial GIS. The ArcSDE technology (in the ArcGIS for Server product) enables end-users to manage enterprise geodatabases in database management systems such as Oracle, DB2, and PostgreSQL, and to perform many useful server-side functions (ESRI, 2014c). ArcSDE stands as a gateway for ArcGIS Desktop and other products accessing the database management systems (Figure 5.9).



**Figure 5.9:** ArcSDE as a gateway to DBMSs

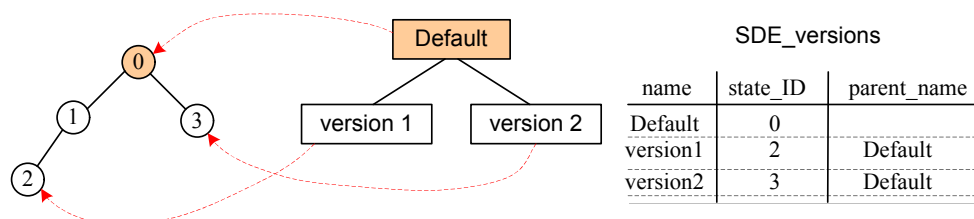
The *database version manager* module of ArcSDE called versioning provides an editing environment for ArcGIS Desktop and other applications. This environment supports concurrent multi-user editing without creating multiple copies of the data. To manage the versions, ArcSDE uses the concepts of *state* and *state tree*. A state is a container for changes to the database. The hierarchical state tree determines the lineage of a state that determines a specific state of the database. The tables `SDE_STATES` and `SDE_STATE_LINEAGE` store states and the state tree of the database (Figure 5.10).



**Figure 5.10:** ArcSDE states and the state tree

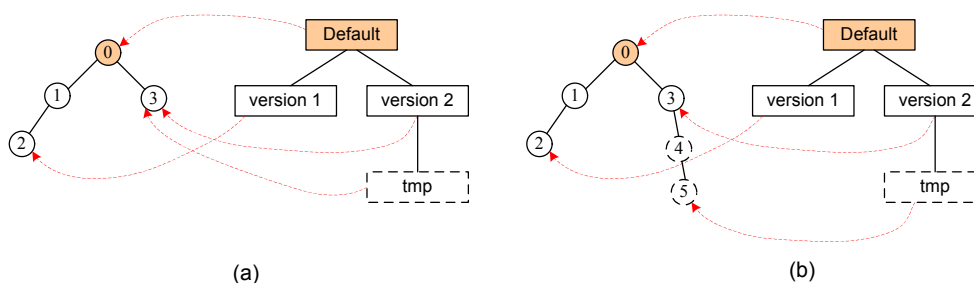
Each version refers to a specific state, thus referring to the lineage that determines a specific state of the database. Multiple versions may point to the same

state if desired, and over time, versions will move from one state to another. The SDE\_VERSIONS table stores information about the version tree and its references to the state tree. Figure 5.11 shows an example of a version tree.



**Figure 5.11:** ArcSDE version tree

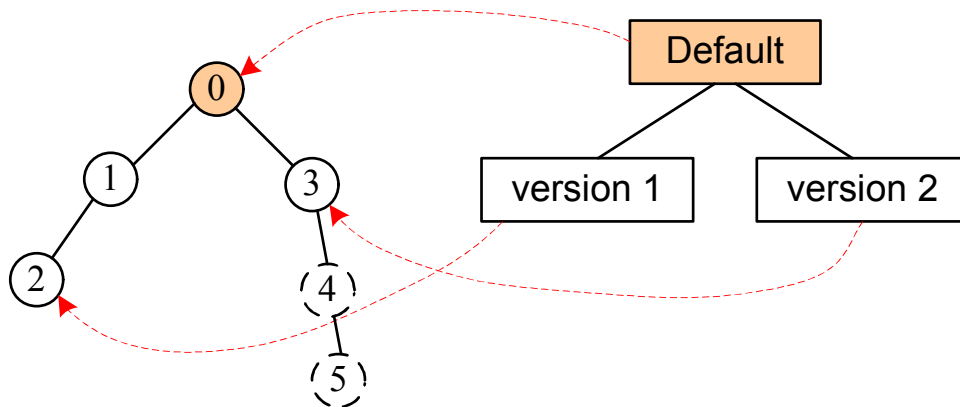
In ArcGIS desktop, a standard editing workflow starts with the start editing command, is followed by the zero or certain *save editing* commands, and finally ends with the *stop editing* command. The *start editing* command registers a unique editing session of a user (editing user) and in a version (editing version). When starting an editing session, a template version is created whose parent is the editing version, e.g., version *tmp* in Figure 5.12a. Each change, such as creating a new feature, deleting an existing feature, or updating an existing feature, is marked by a new state. Figure 5.12b shows versions and states after two changes. When the save editing command is performed in the editing session, the pointers of the editing version are changed to refer to the state referenced by the template version. If the stop editing command is performed without saving, all of the states created in the session are removed. The template version is not explicitly stored in the database, and it is removed from the memory when the editing session stops.



**Figure 5.12:** Versions and states in an editing session; (a) in a starting session; (b) after two edits

In the case of system crashes, e.g., the editing computer is powered off, the versions and states of the database may be in a configuration, such as that in

Figure 5.13. ArcSDE provides a tool, named *Compress*, which removes all of the unreferenced states that may have occurred in the above situation.



**Figure 5.13:** An example of the database after a system crash

ArcSDE manages the versioned and un-versioned tables. A versioned table must have an integer primary key, normally named *objectid*, to identify a row or a feature in the table. This table must also be registered to the database that stored information in the `SDE_TABLE_REGISTRY` table. During registration, two new system table, called *delta* tables, are created to store changes to the table. The names of two delta tables are in the following convention. The name of the *Adds* table, for insert and update commands, and the name of the *Deletes* table, for update and delete commands, start with the character *a* and *d*, respectively, and the postfix is the registration number of the table, e.g., *a10* and *d10*. Once registered as versioned, the table participates in all of the versions of the database. The data structure for versioning in ArcSDE is shown in Figure 5.14.

In Figure 5.14, the `SDE_MVTABLES_MODIFIED` table stores the information of which tables have been modified in a state. The information in this table will improve the performance of queries. Optionally, a versioned view of a versioned table can be created. The versioned view reconstructs the contents of the versioned table for a specified version of the database. Figure 5.15 shows the versioned view of the *test1* table (*registrationid* is 7).

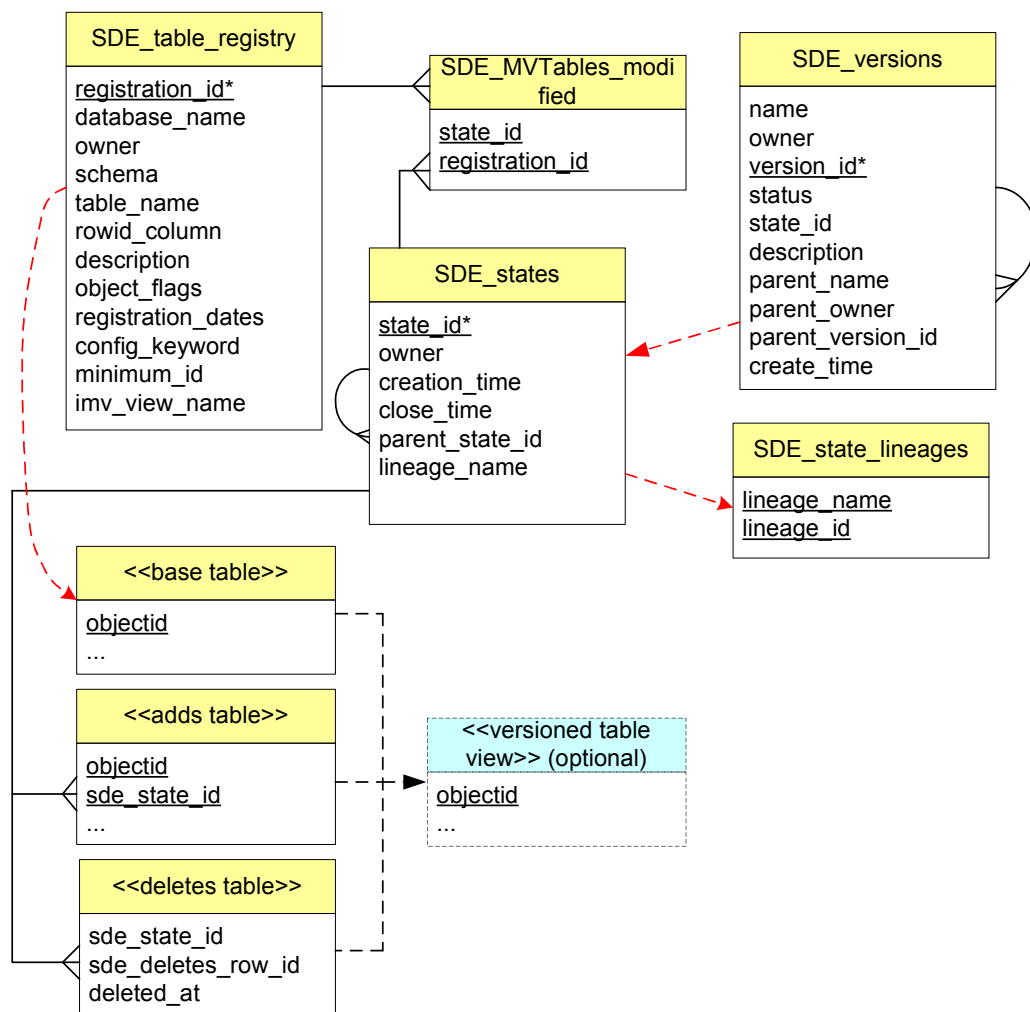


Figure 5.14: ArcSDE versioning data schema

## 5.4 TGSIS database versioning and Oracle Workspace Manager

TGSIS database versioning (Le et al., 2014 (accepted)) uses the ideas from Oracle Workspace Manager (OWM) (Oracle, 2013a,b) but is tailored to geoscience applications in which geological objects, such as surfaces, are considered. TGSIS database versioning is implemented in the PostgreSQL system (PostgreSQL, 2014).

```

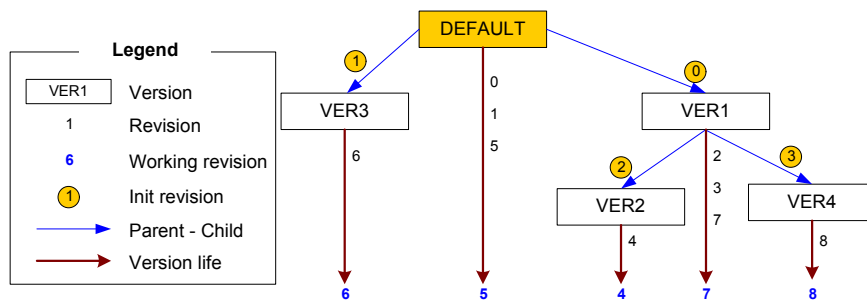
-----
CREATE OR REPLACE VIEW sde.test1_view AS
  SELECT b.objectid, b.porosity, 0 AS sde_state_id, b.shape
    FROM test1 b
   WHERE NOT (EXISTS ( SELECT d.sde_deletes_row_id, d.sde_state_id
                        FROM d7 d
                       WHERE d.sde_state_id = 0
                           AND sde_in_current_lineage(d.deleted_at) = true
                           AND b.objectid = d.sde_deletes_row_id))
UNION ALL
  SELECT a.objectid, a.porosity, a.sde_state_id, a.shape
    FROM a7 a
   WHERE NOT (EXISTS ( SELECT d.sde_deletes_row_id, d.sde_state_id
                        FROM d7 d
                       WHERE sde_in_current_lineage(d.deleted_at) = true
                           AND a.objectid = d.sde_deletes_row_id
                           AND a.sde_state_id = d.sde_state_id
                           AND sde_in_current_lineage(a.sde_state_id) = true))
                           AND sde_in_current_lineage(a.sde_state_id) = true;
-----

```

**Figure 5.15:** An ArcSDE versioned view

### 5.4.1 Data structure and functions

Like OWM, TGSIS database versioning manages revised data at the row level, i.e., only changed rows of an object are updated. For example, when a triangulated mesh is changed at some vertices, only the changed vertices and its corresponding triangles (changed rows) are updated; unchanged vertices and unchanged triangles are left unchanged. An identifier number, i.e., revision, is assigned to revised data to define the version the revised data belong to. Figure 5.16 shows the relationship between version and revision.



**Figure 5.16:** Version and revision

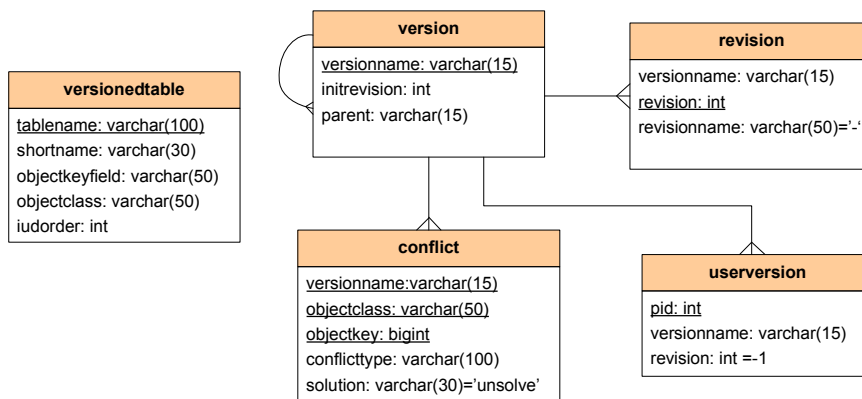
Version DEFAULT and revision 0 are the initial values of the system and cannot be changed or removed by end-users/applications. Version DEFAULT is the ancestor of any other version. Each version, except DEFAULT, has a parent version and may have child versions. A version has a set of revisions and its working revision, which is the maximum element of its revision set. A version has an initial revision, except DEFAULT. Revision is an integer number whose value is obtained from an object sequence, REVISION\_SEQ, whose starting value is 1 with

an increment of 1 for each call of a function named *nextval*(REVISION\_SEQ). The diagram in Figure 5.16 is a result of the following action sequence. (1) At system initialization, version DEFAULT and revision 0 have been set up. (2) Version VER1 has been created. Because, at this time, the working revision of DEFAULT was 0, the initial revision of VER1 is 0. After creating VER1, the revision set of DEFAULT is {0, 1}, the revision set of VER1 is {2}, the working revision of DEFAULT is 1, and the working revision of VER1 is 2. (3) Version VER2 has been created as a child of VER1; the initial revision of VER2 is 2 and the revision sets of DEFAULT, VER1, VER2 are {0, 1}, {2, 3}, and {4}, respectively. (4) Version VER3 has been created as a child of DEFAULT; its initial revision is 1 because the working revision of DEFAULT is 1; the revision sets of DEFAULT, VER1, VER2, VER3 are {0, 1, 5}, {2, 3}, {4}, {6}, respectively. (5) Version VER4 has been created as a child of VER1; its initial revision is 3; the revision sets of DEFAULT, VER1, VER2, VER3, VER4 are {0, 1, 5}, {2, 3, 7}, {4}, {6}, {8}, respectively; and their working revisions are 5, 7, 4, 6, 8, respectively.

When end-users/applications work with the database, they are working in a specific version (working version), which is DEFAULT when they do not explicitly select any. Each modification action, i.e., insert, update, delete, uses the working revision and the corresponding version. Database queries use the working revision set to define which data can be viewed on a version. For example, when working in VER3, all data are data belonging to working revision set {6, 1, 0}. End-users/applications can also select a revision to retrieve the data of interest, i.e., a viewing revision, which is not necessarily the working revision. For example (see Figure 5.16), after going to VER1, revision 3, the viewing revision is now 3, but the working revision is 7. Even after going to VER4, revision 2, the viewing revision is 2, but the working revision is 8, and the viewing revision set and working revision set are {2, 0} and {8, 3, 2, 0}, respectively.

To store the information about versions, revisions, the working version of an end-user, the versioned tables and conflicting data between versions, the core database schema has been designed as shown in Figure 5.17.

Two relations, VERSION and REVISION, contain all data, e.g., those exemplified in Figure 5.16. End-users/applications can create a revision with its name to mark a snapshot point in the database and, therefore, can view data at this point at any time. Attribute *revisionname* stores the name of a revision. Relation USERVERSION stores the data from working end-users including process iden-



**Figure 5.17:** Core data schema

tifiers (*pid*), working versions (*versionname*) and viewing revisions (*revision*). If the viewing revision is not defined, the working revision of the working version is used instead. The viewing revision, which differs from the working revision of the working version, is only used for queries.

Initial values of VERSION and REVISION are inserted using the two following commands.

---

```

insert into VERSION(versionname='DEFAULT', intirevision=0,
                    parent=' ');
insert into REVISION(versionname='DEFAULT', revision=0,
                    revisionname='- ');
  
```

---

In TGSIS, end-users/applications see data as objects or features organized by object/feature classes, which are a set of related tables. For example, geological surfaces can be managed in a database as the surface class with three tables: VRTX, TRGL, and SURFACE, as shown in Sub-section 5.4.2. A surface is identified by a key, called the object identifier. A surface can be seen with different shapes on different versions, and each representation on a version is called a version of that surface. When a surface has been changed on a version and on the parent version since the version was created, the surface is a conflicting object on the version. This conflict may come from either table VRTX, TRGL, SURFACE, or all of them. The names of these tables are called conflicting parts of the surface. In this case, the version is a conflicting version and cannot merge with its parent version. Conflict solving actions, which will be presented in Section 5.6,



tablename	shortname	objectkeyfield	objectclass	iudorder
postgst.surface_ver	surface	geoid	surface	0
postgst.vrtx_ver	vrtx	geoid	surface	1
postgst.trgl_ver	trgl	geoid	surface	2

**Table 5.1:** Sample data from the VERSIONEDTABLE table

need to be performed before merging conflicting versions.

Relation VERSIONEDTABLE contains the name of the table (*tablename*), the short name to notify the table of containing conflicts (*shortname*), the name of key field of objects (*objectkeyfield*), the class of objects (*objectclass*) and the number assigned to each versioned table in the set of the versioned tables of a class (*iudorder*). The numbers in the attribute *iudorder* are used to control the order of insert, update, and delete actions to obey foreign key constraints of versioned tables of an object class. For example, in the database managing geological surfaces, three tables SURFACE, VRTX, TRGL need to be versioned. Data in relation VERSIONEDTABLE are shown in Table 5.1.

In the relation CONFLICT, the type of conflict (*conflicttype*) is the information about the conflicting object and identifies the short name of the conflicting tables. The value “*vrtx*”, for example, means that two versions of the surface conflict at some vertex. The solution for each conflicting object is defined by end-users. The values of each solution may be “*origin*”, “*current*” or “*unsolved*”, meaning that the original or current version should be used, or that the conflict is unsolved, respectively.

For application tables, each versioned application table has been renamed to the new name, which is the old name and postfix “\_VER”. A new database view with the old name is then added. Two columns, REVISION (int), LASTREVS (int[]), are added to the table. The column *revision* is also added to the primary keys of the table. Metadata from the application table must be registered in VERSIONEDTABLE. Queries and modification commands on table *R* are redefined as the following.

#### **Selection:**

$$\sigma(R) = \sigma_{revision \in getRevisions(true) \wedge (lastrevs \cap getRevisions(true)) = \emptyset}(R\_VER);$$

**Insertion:**

*Insert into R = Insert into R\_VER(revision = getWorkingRevision(),  
lastrevs = '{'});*

**Update:**

```

Update Rid=mid = {
  if ( $\sigma_{id=m_{id} \wedge revision=getWorkingRevision()}(R\_VER) \neq \emptyset$ )
    Update Rid=mid;
  else {
    Update R_VERid=mid ∧ revision ∈ getRevisions() ∧ (lastrevs ∩ getRevisions()) = ∅
      (lastrevs = lastrevs ∪ getWorkingRevision());
    Insert into R_VER(Rid=mid, revision = getWorkingRevision(),
      lastrevs = '{'});
  }
}

```

**Delete:**

```

Delete Rid=mid = {
  if ( $\sigma_{id=m_{id} \wedge revision=getWorkingRevision()}(R\_VER) \neq \emptyset$ )
    Delete R_VERid=mid ∧ revision=getWorkingRevision();
  else
    Update R_VERid=mid ∧ revision ∈ getRevisions() ∧ (lastrevs ∩ getRevisions()) = ∅
      (lastrevs = lastrevs ∪ getWorkingRevision());
}

```

The following example will explain the previously discussed functions, which will be implemented as a database view and three database triggers for each versioned table. Assume we are working on version VER1 before creating version VER4. At that time, the working revision of VER1 is 3. An insert command to table *R* will create a row with the values of fields REVISION and LASTREVS, which are 3 and  $\emptyset$ , respectively. If an update or delete command is performed at this time, the row is permanently updated or removed from the data, as is performed in normal databases. Now, we create version VER4 (see Figure 5.16). The working revision of VER1 is 7 and the working revision of VER4 is 8. If we work on VER1 and update the row (the old row has values of fields REVISION and LASTREVS of 3 and {7}, respectively), a new row is added and values of its fields REVISION and LASTREVS are 7 and  $\emptyset$ , respectively. If we work on VER1 and delete the row, the row is not removed from the database (because it is being existing in version VER4), and the values of fields REVISION and LASTREVS are 3 and {7}, respectively. If we work on VER4 and delete the row, the values of fields REVISION and LASTREVS are 3 and {8}, respectively.

### 5.4.2 Tailoring for geological surfaces

The data schema for geological surfaces is extended to give TGSIS the ability to maintain different versions of geological surfaces, and therefore support long transactions. Figure 5.18 shows both the non-versioned schema as well as the versioned schema of a geological surface database.

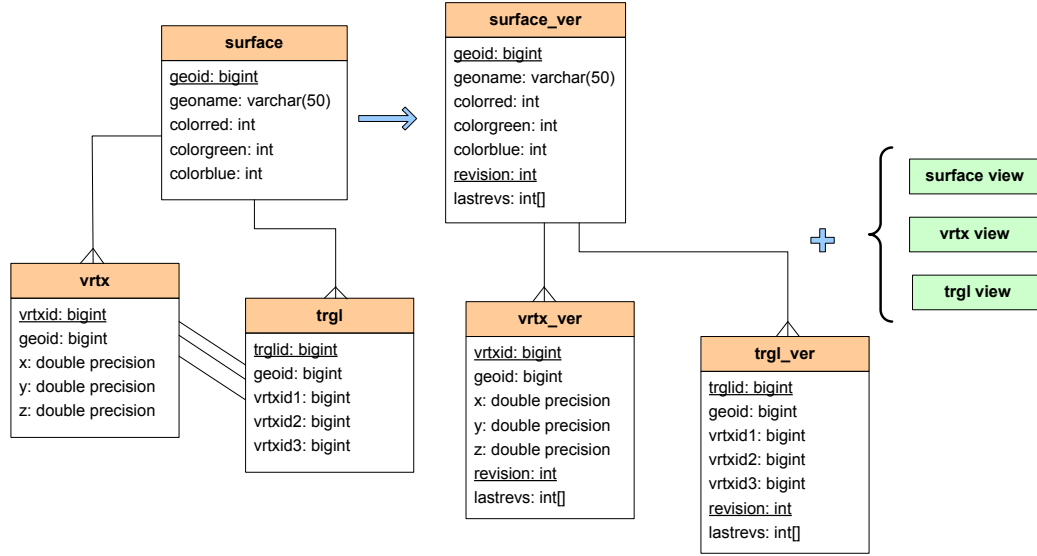


Figure 5.18: Surface schemata

Note that we remove the associations between *vrtx\_ver* and *trgl\_ver* so that a *trgl\_version* row can refer to three *vrtx\_ver* rows in different versions.

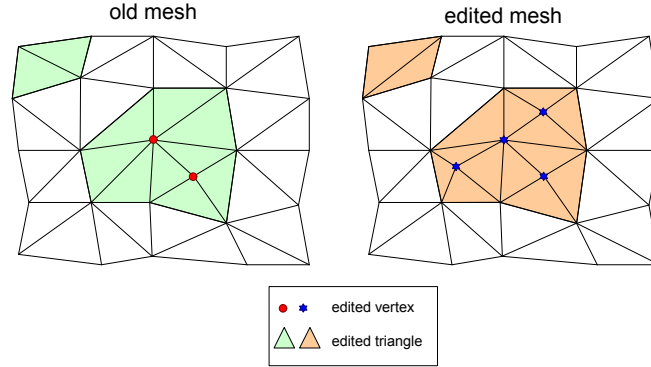
## 5.5 TGSIS mesh comparison

Mesh comparison, in this study, means finding the parts (vertices, edges, and triangles in case of triangulated meshes) where the two input meshes differ. This issue can be performed by a nearest neighbor search, range search, and point location. Our algorithm is simpler because it is defined in 3-dimensional Euclidean space (low dimensionality) using Euclidean distance, and we only need to find a coincident vertex. Hundreds of publications from researchers in a number of fields are related to nearest neighbor searching, range searching and point location issues (Clarkson, 2006, Dhanabal and Chandramathi, 2011). In general, a searching algorithm includes two phases: preprocessing and querying. In the preprocessing phase, some data structures are used to accelerate the query phase. The most

famous among these data structures are, for example, octree, k-d tree, AABB tree, locality-sensitive hashing and many modifications of these (Bentley, 1975, CGAL, 2014, Datar et al., 2004, Liaw et al., 2010, Samet, 1994, Zatloukal et al., 2002). Until now, to the best of our knowledge, for a point set in 3-dimensional Euclidean space and Euclidean distance, the best solutions have, on average, the complexity of  $O(\log(n))$  querying time (for each query), while preprocessing time and space complexities are  $O(n\log(n))$ . The TGSIS mesh comparison algorithm, which uses the k-d tree, queries in time  $O(\log(n))$  in the worst case (for each query), preprocesses in time  $O(n\log(n))$ , and uses space  $O(n)$ . In the rest of this subsection, we will review the TGSIS mesh comparison.

In the sense of TGSIS, object comparison aims to compare two representations of an object. A representation exists in the database and is called the old representation. Another representation exists in the memory and is called the edited representation. The object comparison algorithm identifies the parts of the old representation which need to be removed from the database and the parts of the edited representation which need to be added to the database so the revision is stored in the database. For example, when objects are triangulated meshes, the mesh comparison identifies vertices and triangles of the old mesh that need to be removed from the database. These sets are called the set of edited vertices and the set of edited triangles of the old mesh. The complements of these sets are called the set of non-edited vertices and the set of non-edited triangles of the old mesh. The mesh comparison also identifies vertices and triangles of the edited mesh that need to be added to the database. These sets are called the set of edited vertices and the set of edited triangles of the edited mesh. Their complements are called the set of non-edited vertices and the set of non-edited triangles of the edited mesh. A non-edited vertex is identified by comparing geometrical coordinates, while assuming that the old mesh and the edited mesh are normal meshes, i.e., there are no two coincident vertices and no two coincident triangles in a mesh. A vertex of the edited mesh is a non-edited vertex if a vertex of the old mesh exists and is coincident to it. Additionally, the coincident vertex is a non-edited vertex of the old mesh. Because a triangle is represented by its three vertices, a non-edited triangle of the old mesh or the edited mesh is a triangle whose three vertices are non-edited, and a corresponding triangle of the edited mesh or the old mesh exists, respectively. In Figure 5.19, for example, the triangle in the top left of the edited mesh, in the result of a flip action, has three non-edited vertices

but no corresponding triangle of the old mesh, so it is a edited triangle.



**Figure 5.19:** Mesh comparison

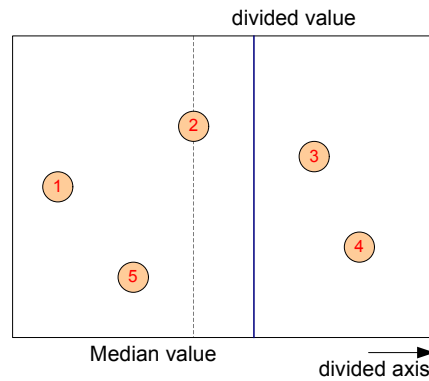
By using a data structure that allows us to easily access incident triangles from a vertex, once a coincident vertex pair of the old mesh and the revised mesh has been identified, two lists of their incident triangles are compared to infer the non-edited triangles. On average, each vertex has 6 incident triangles, meaning there are approximately 36 comparisons of triangle pairs. Non-edited vertices can be identified by solving the following problem.

**Problem 5.1** *Given two point sets in 3-dimensional Euclidean space, named the source point set and the target point set, find a subset of the source point set such that (1) each point of it is coincident to a point of the target point set and (2) it has a maximum length.*

The naive solution to this issue is sequentially comparing each pair of vertices. The complexity of this solution is  $O(n^2)$  for time and  $O(n)$  for space, when two point sets have nearly the same number of points,  $n$ . Our algorithm identifies non-revised vertices in time  $O(n \log(n))$  and space  $O(n)$  using two steps.

In the preprocessing step, the vertex set of the edited mesh is built as a k-d tree using a so-called sliding-median-widest-spread strategy. The widest spread axis is divided using the median value as the division value. If it is coincident to some vertices, we slide that value so that it is a mid-value of its nearest vertex, but not coincident along divided axis, and the median value (see Figure 5.20).

Because this k-d tree is of  $\log(n)$  depth, the coincident querying from the root node down to the leaf node in the worst case is  $\log(n)$ . To construct k-d tree of the edited mesh, we first sort the vertices of the edited mesh on each of



**Figure 5.20:** An example of k-d tree sliding-median-widest-spread

the dimensions,  $x, y, z$ , separately. This task can be done in time  $O(n \log(n))$  and space  $O(n)$ . We then find the widest spread axis, the median value and the divided value. The complexity of constructing a k-d tree is in time  $O(n \log(n))$  and space  $O(n)$  (De Berg et al., 2000). The algorithm to construct k-d tree is implemented as the construction of the k-dtree class and the algorithm to query a coincident point is implemented as the function of this class, named *findCoincidence*. This class is used in the *CompareMeshes* procedure.

The algorithm is implemented in the *CompareMeshes* procedure whose input is the edited mesh and the old mesh. The outputs of this procedure are two sets, the vertex set and the triangle set of the old mesh, which need to be removed. The edited mesh was modified in the field *dinfo* of its vertices and triangles. A vertex and triangle whose *dinfos* equal *ExistInDB* are a non-edited vertex and triangle, respectively. A vertex and triangle whose *dinfos* equal *NonExistInDB* need to be inserted into the database.

## 5.6 Version merging strategy

In the database version manager, a version can be merged with its parent version. An object in a version  $A$  is a conflicting object if it was changed in version  $A$  and in the parent version of version  $A$  since version  $A$  was created. In the merging process, three options are suggested for each conflicting object.

1. Select current version
2. Select parent version
3. Manually solve (with some helping tools)

Manually solving a conflict mesh means merging the current mesh and the parent mesh of an object. This task can be completed using the morphological interpolation method for intermediate geometries to calculate the average of two meshes (see (Le, 2013)).

## 5.7 Summary

A long transaction is a feature extended to traditional database management systems. Long transactions, for example, isolate a group of long duration changes, store importance states of the database persistently, support analyses by “what-if” scenarios, prevent the lost-update phenomenon. Workspace manager or database versioning is the basis of long transactions. ArcGIS/ArcSDE versioning creates a template version for each editing session in ArcGIS/ArcMap, persistently saves any user interactive change to the database, and reconciles data when closing the editing session. Therefore, ArcGIS/ArcSDE along with ArcGIS/ArcMap provide a high performance for concurrent multi-user editing. Oracle Workspace Manager and TGSIS provide the long transaction feature for general purpose SQL applications. OWM also supports temporal data with the valid-time. Currently, TGSIS database versioning is implemented as a prototype in the PostgreSQL without privilege and lock management.

The TGSIS database versioning is an extension to the PostgreSQL system. It enables applications to manage data versions without code changing. Currently, the TGSIS database versioning, equipped with the TGSIS mesh comparison and the version merging strategy, can be used to manage effectively many versions of structural geological models, e.g., geological models which are frequently updated to reflect new surveyed data during a mining excavation.





# Chapter 6:

## Prototype software

---

### 6.1 Introduction

A spatio-temporal geoscience information system (TGSIS) was built as a prototype software. TGSIS implemented the theories and algorithms mentioned in the previous chapters. TGSIS includes the following three modules: TGSIS server module, TGSIS Manager, and TGSIS gOcad plugin (Paradigm-GOCAD, 2014). The second and the third data models presented in Chapter 3 were set up in the TGSIS server module, which used the PostgreSQL system (PostgreSQL, 2014) as a database management system.

The algorithms of the TGSIS morphological interpolation were implemented in the TGSIS gOcad plugin module. The software uses the Boost C++ libraries (Boost, 2014) and the Computational Geometry Algorithms Library (CGAL, 2014). To use CGAL parameterisation when the parameterisation domain is a unit regular  $k$ -polygon, a new class implementing the *BorderParameterizer\_3* concept was implemented. The software also uses the OpenNL library (OpenNL, 2014) integrated with CGAL as a solver for sparse linear equations to find the trajectories. The software provides a graphical user interface for the end-user to input and modify the constraints. The boost graph library is used to find the shortest path to construct cutting and fence paths.

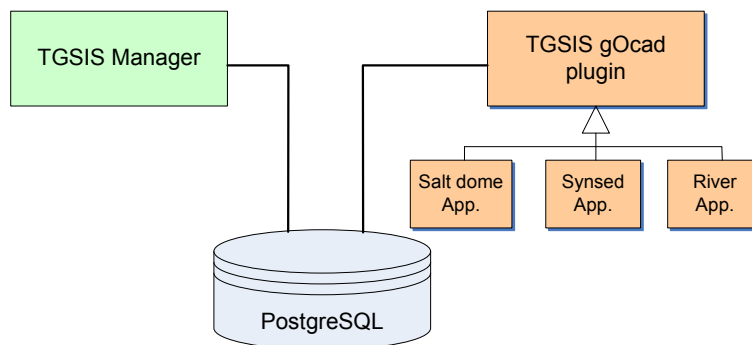
Functions, triggers, and views were also set up to implement algorithms for long transactions and the database versioning (Chapter 5). The algorithm comparing

two triangulated meshes presented in Chapter 5 uses the gOcad API library.

In this chapter, Section 6.2 presents the architecture of the prototype software. Three modules *TGSIS server*, *TGSIS Manager*, and *TGSIS gOcad plugin*, are briefly described in Sections 6.3, 6.4, 6.5, respectively.

## 6.2 Architecture

TGSIS was built using a client–server architecture. On the server side, the database management system is PostgreSQL; the data schema shown in Chapter 3 is set up, and stored–procedures and triggers are implemented. On the client side, there are two software modules, namely, the *TGSIS Manager* and the *TGSIS gOcad plugin* (Paradigm-GOCAD, 2014). The TGSIS Manager is primarily used by the application administrators who are responsible for determining which data should be managed and the structures of these data. TGSIS Manager also provides certain functions for quick querying data in the database. The TGSIS gOcad plugin is a software module developed as a gOcad plugin and operates in the environment of the gOcad system. Using this architecture, the TGSIS gOcad plugin has all of the functions of the geomodeling from gOcad and all of its plugins. The TGSIS gOcad plugin is primarily used by the application operators, who are responsible for capturing, storing, manipulating, querying, and reporting data in the database, i.e., performing the select, insert, update, and delete operations on the database. Figure 6.1 shows the architecture of TGSIS.



**Figure 6.1:** The architecture of TGSIS

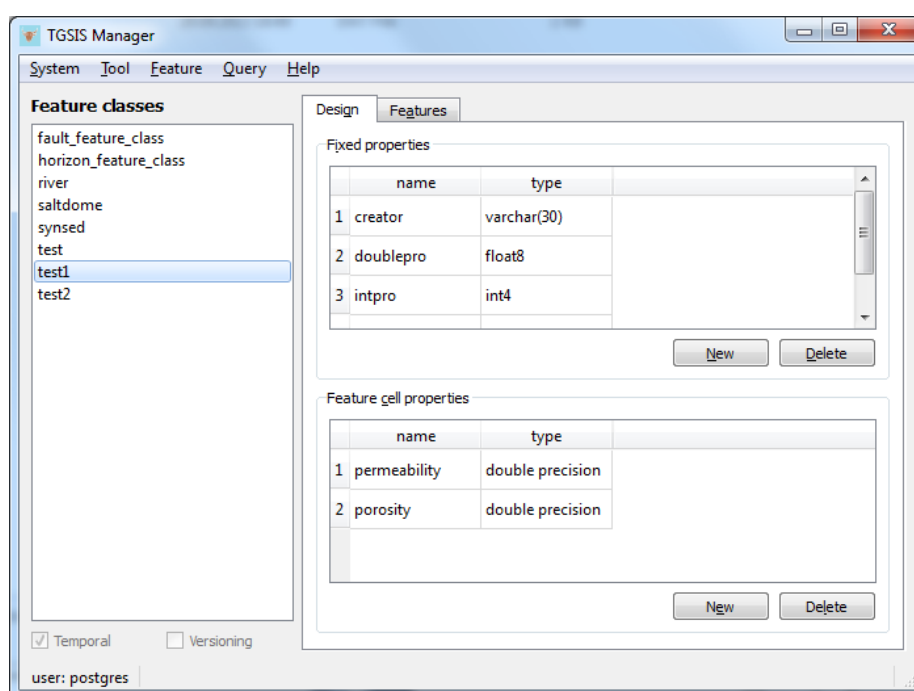
Applications of TGSIS are typically but not necessarily gOcad plugins that are inherited from the TGSIS gOcad plugin. Consequently, the applications can use or extend all of the functions of the TGSIS gOcad plugin.

## 6.3 Server side – PostgreSQL

PostgreSQL is an open-source object-relational database management system. The source code of PostgreSQL is in the C language. User functions running inside PostgreSQL can be written in certain languages, such as SQL, C, PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python. Other procedural languages can also be used in PostgreSQL using the appropriate methods. C was used to develop the input/output operations for the TGSIS system. Because PL/pgSQL is a simple and flexible language, it was chosen to write procedures and triggers for database versioning, as shown in Chapter 5.

## 6.4 TGSIS Manager

The TGSIS Manager is desktop software. The main graphical user interface (GUI) of this software is shown in Figure 6.2.

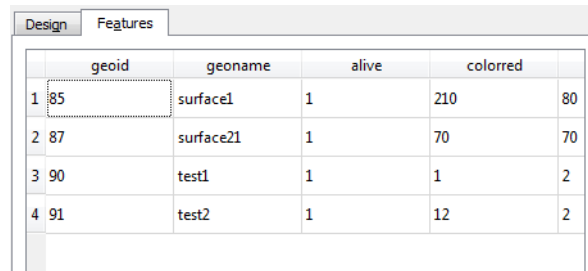


**Figure 6.2:** The graphical user interface of TGSIS Manager

The *feature classes* list shows all of the feature classes defined in the system. Each feature class is designed with some *fixed properties*, i.e., properties that are attached to the whole feature and do not change with time, and some *feature cell*

*properties*, i.e., properties that are attached to vertices or cells of a feature in the feature class. Administrators can select each property attached to either vertices or cells when creating a new feature class. A feature class can be temporal with or without versioning.

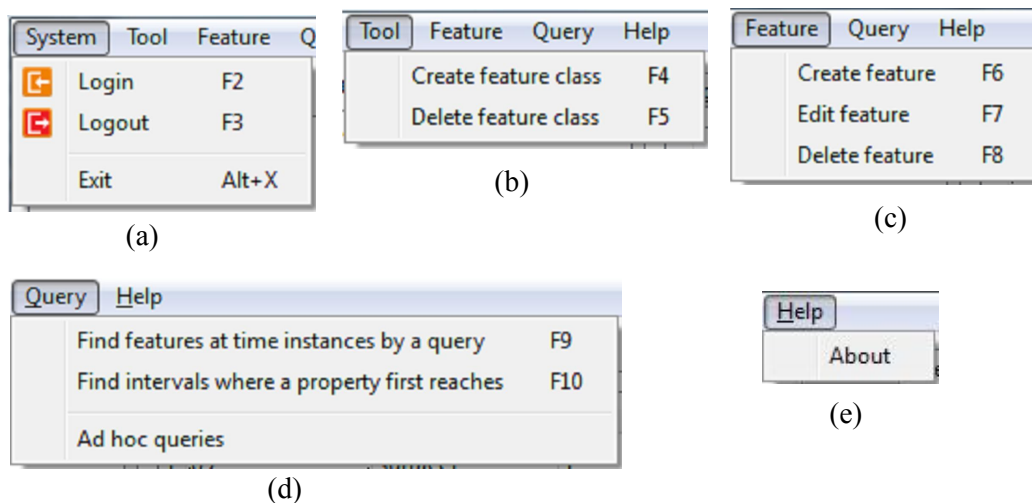
The *features* tab in the main GUI shows all of the existing features of a selected feature class (Figure 6.3).



	geoid	geoname	alive	colored	
1	85	surface1	1	210	80
2	87	surface21	1	70	70
3	90	test1	1	1	2
4	91	test2	1	12	2

**Figure 6.3:** The list of features in a feature class

Figure 6.4 shows the menus of the TGSIS Manager.



**Figure 6.4:** Menus of the TGSIS Manager

To login, logout, or exit the system, use the *system* menu. To create or delete a feature class, use the *tool* menu. Select a feature class and then use commands in the *feature* menu to create, edit, or delete a feature in the feature class. Figure 6.5 shows the dialog for editing features.

We can determine the following: (i) all of the features and their existing time instances by certain given properties (Figure 6.6), (ii) all of the features and their existing time intervals or time points at which a property of a feature first reaches

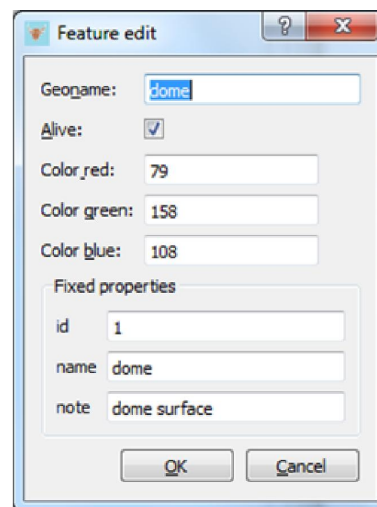


Figure 6.5: Dialog for editing features

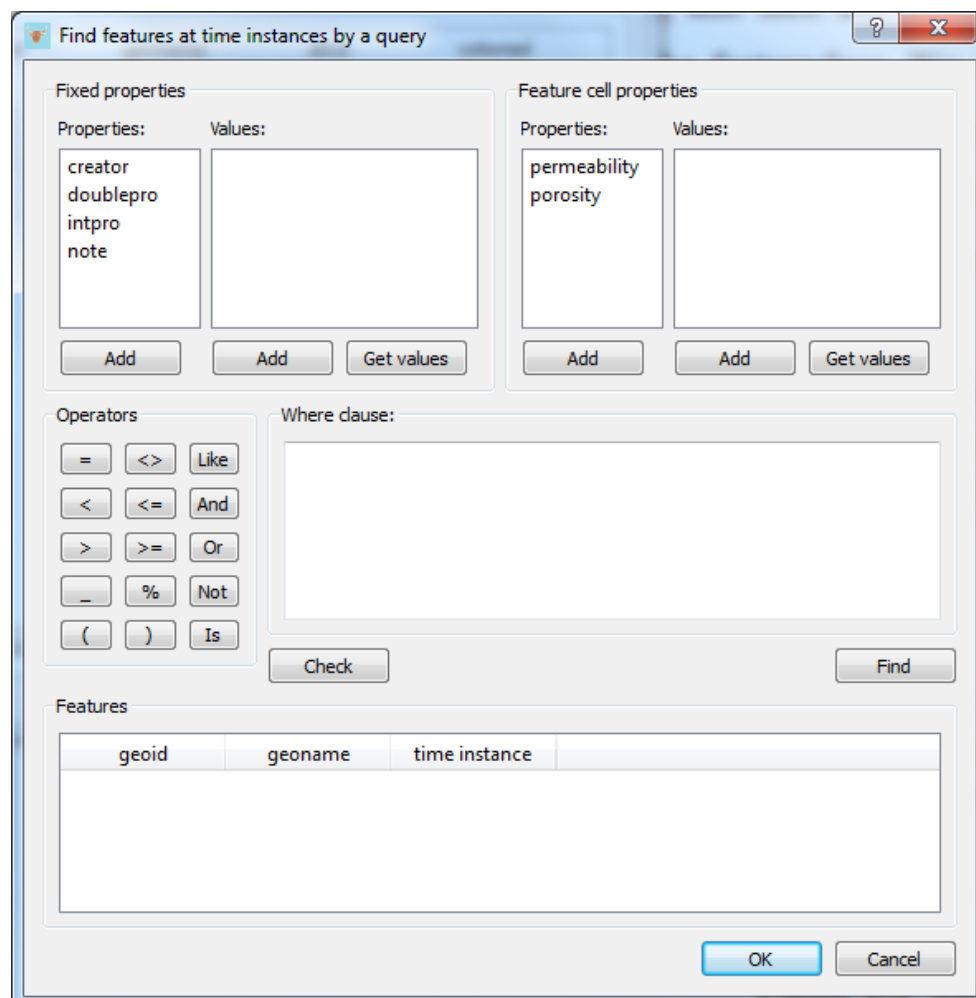
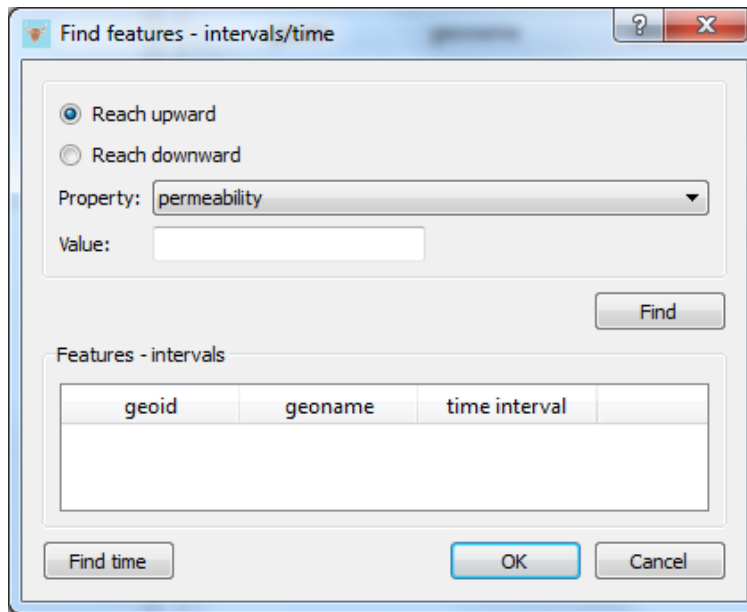


Figure 6.6: Find features and their existing time instances by certain given properties



**Figure 6.7:** Find features and intervals or time points

while the feature evolves in time (Figure 6.7), and (iii) certain interests with the ad hoc queries. The query menu provides these tools for querying. The last menu is *help* menu.

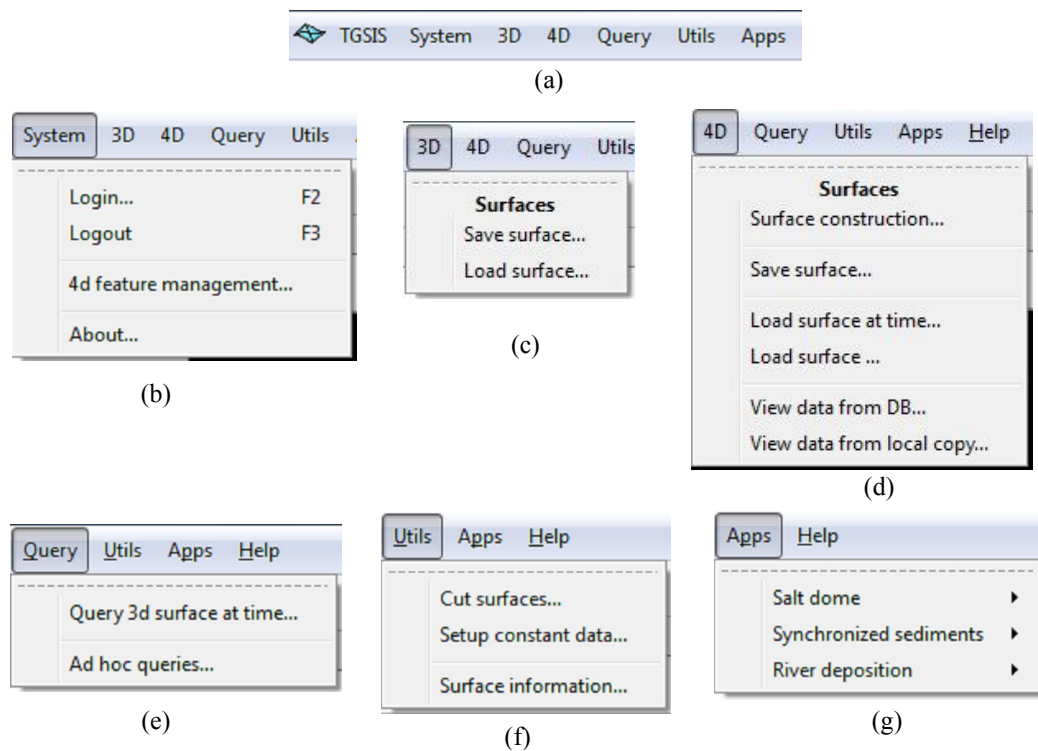
## 6.5 TGSIS gOcad plugin

The TGSIS gOcad plugin is primarily used by operators responsible for data manipulation. The menus of TGSIS are shown in Figure 6.8.

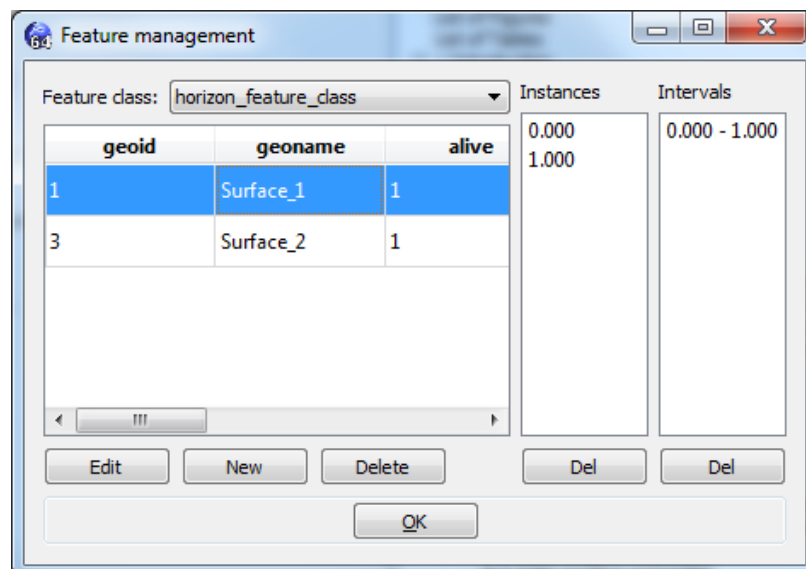
The goal of the *Feature management* function in the *system* menu is to manage the features. We can create a new feature, update certain properties of a feature, or delete a feature. We can also view and remove an instance or a time interval of the selected feature. Figure 6.9 shows the graphical user interface of this function.

The functions in the *3D* menu are used to save and load surfaces that are instances of certain features at any time instance. When saving a surface into the database, the time, at which the surface represents the object, is given (Figure 6.10a). Properties of gOcad surface are stored into appropriate database properties using a mapping (Figure 6.10b).

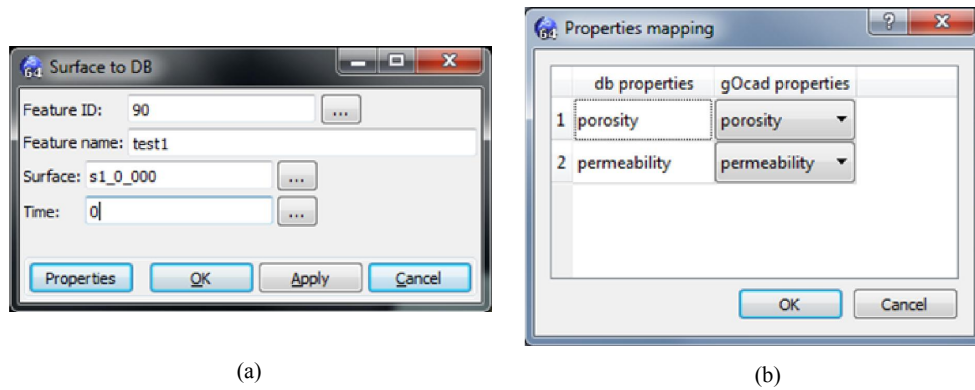
Figure 6.11 shows the GUI of the *load surface* function. Properties can be loaded with the geometry. Time can be stored as a property of the surface in



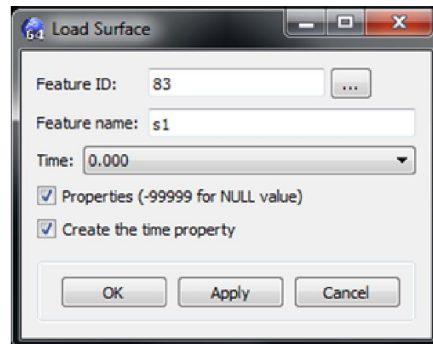
**Figure 6.8:** The main graphical user interface of the TGSIS gOcad plugin



**Figure 6.9:** The GUI of the feature management function



**Figure 6.10:** Saving a surface into the database



**Figure 6.11:** Loading a surface from the database

gOcad or as the name of the surface.

The functions in the *4D* menu are used to manipulate surfaces changing continuously in time. The *surface construction* function is used to model temporal surfaces using the TGSIS morphological interpolation method shown in Chapter 4. To perform this function, first build the model, then set up at least three control vertices; the setup of fence paths or attached constraints are optional steps. Finally, perform the calculation. All of the setup data can be saved in external files for later loading. Figure 6.12 shows the graphical user interface of this function.

The *save surface* function is used to store temporal surface into the database (Figure 6.13a). A mapping is used to map gOcad properties to their appropriate database properties (Figure 6.13b).

The *load surface at time* and *load surface* functions are used to load surface representing the object at a given time and to load temporal surface, respectively. Figure 6.14a and b show the GUI of these functions.



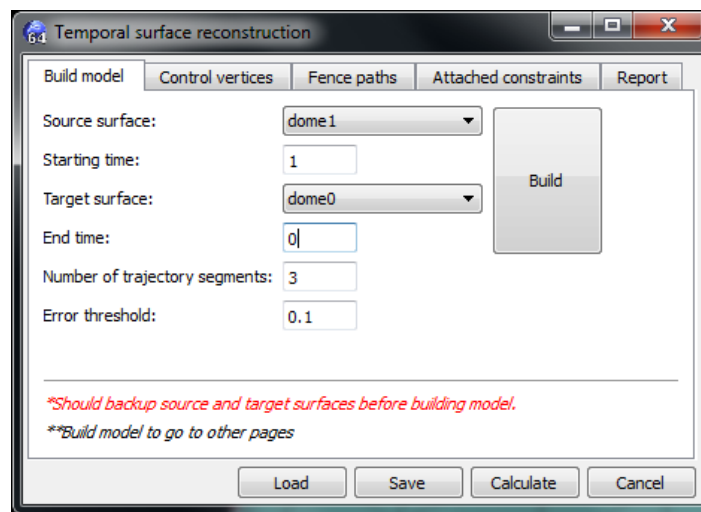


Figure 6.12: The GUI of the temporal surface construction

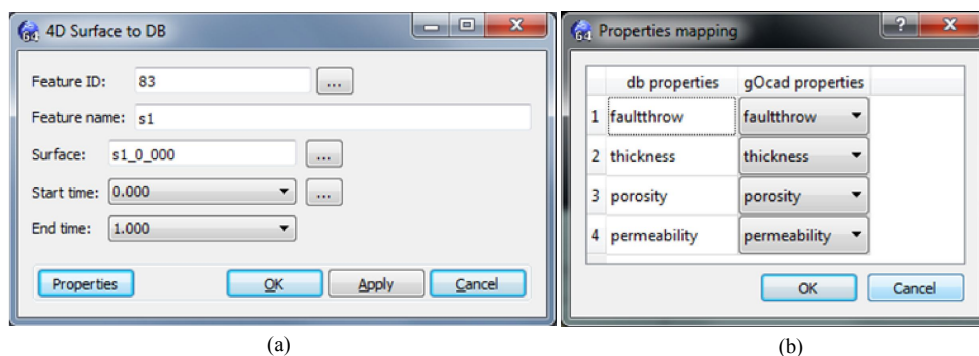


Figure 6.13: The GUI of the saving data function

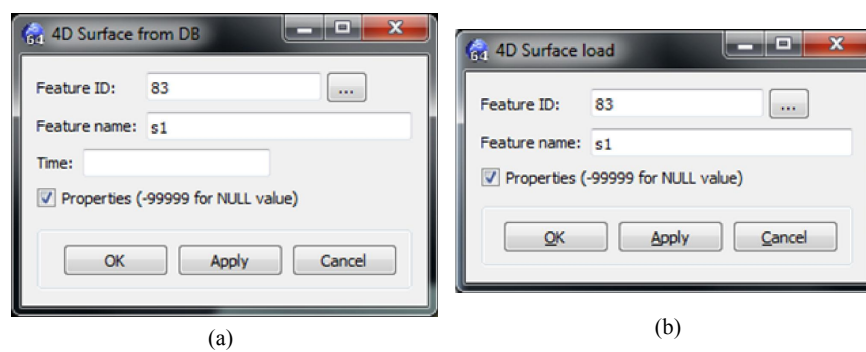
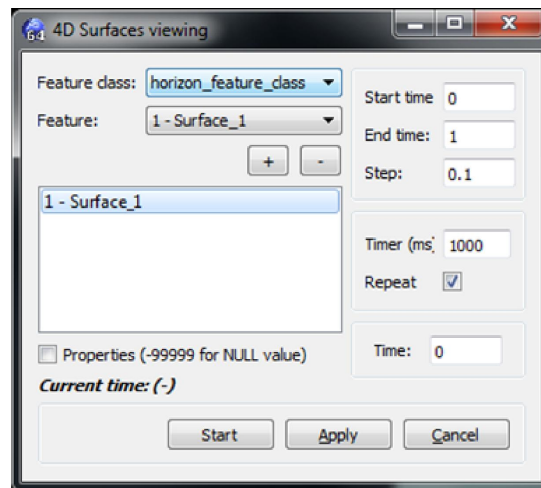
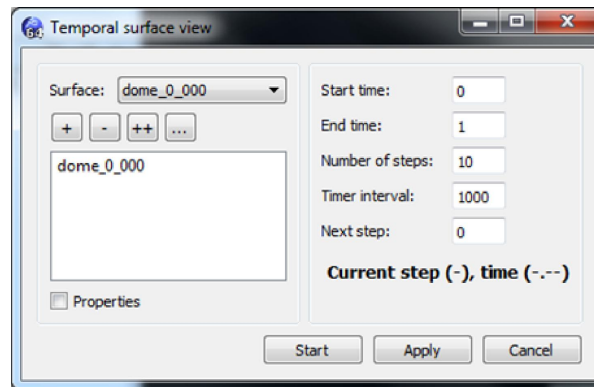


Figure 6.14: The GUI of the loading data function



**Figure 6.15:** The GUI of the function to visualise data from the database



**Figure 6.16:** The GUI of the function to visualise loaded temporal surfaces

We can visualise temporal surface from the database directly using the *view data from DB* function (Figure 6.15) or visualise loaded temporal surface using the *view data from local copy* function (Figure 6.16).

The *query* menu provides functions to determine interesting information from the system. The *cut surface* function in the *utils* menu is to perform the *cutting* procedure shown in Chapter 4. The function *setup constant data* constructs a temporal surface, which is a constant in a given time interval. The *surface information* function is used to show interesting information of a surface. The *apps* menu is the graphical entries to functions of an application.

# Chapter 7:

## Sample applications

---

### 7.1 Introduction

This chapter presents certain sample applications, which are running examples for the prototype of the spatio-temporal geoscience information systems (TG-SIS). Samples in Section 7.2 and Section 7.3 used fictitious data for testing the correctness of the algorithms. The sample shown in Section 7.4 used data from the coal mine project. Section 7.2 presents certain spatio-temporal models of geological processes. A geological structural model with some versions is presented in Section 7.3. Section 7.4 describes the application of TGSIS in the coal mine project.

### 7.2 Modeling geological processes

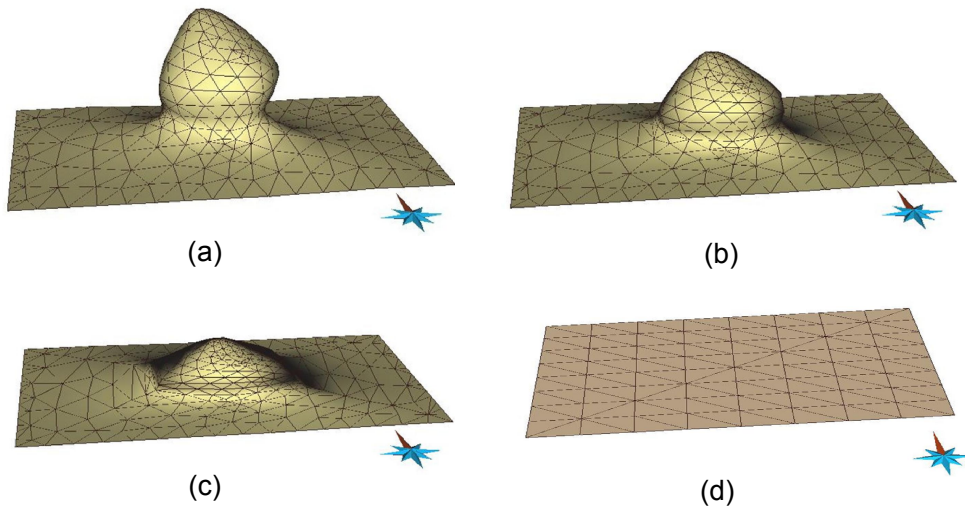
#### 7.2.1 Evolution of the geometry of a salt-dome

In this sample, we modeled the evolution of a salt-dome, which evolves during a gravity driven tectonic process. There are several conceptual models how diapirs grow, such as the Trusheim's model (Trusheim, 1960) and Vendeville and Jackson's model (Vendeville and Jackson, 1992). In this sample, we used our morphological interpolation method to approximate the shapes of the growing salt dome from an undeformed sedimentary salt unit. This morphological in-

terpolation does not reconstruct a tectonic process but can aid in reconstructing changes in geometry of the salt body. The salt body was modeled by its boundary surface, which continuously changes its shape during diapir growth.

The current stage of the salt-dome was given as a triangulated mesh (at time  $t = 1$ ), see Figure 7.1a, and the initial stage of this salt-dome was assumed a flat triangulated mesh (at time  $t = 0$ ), see Figure 7.1d. Four control points at the corners of the mesh at time  $t = 1$  and four corresponding control points at the corners of the mesh at time  $t = 0$  were given. The software created spatio-temporal model which represents intermediate stages of the salt-dome at times between 0 and 1. Figure 7.1b and Figure 7.1c show two intermediate stages of the salt-dome at time  $t = 2/3$  and  $t = 1/2$ , respectively.

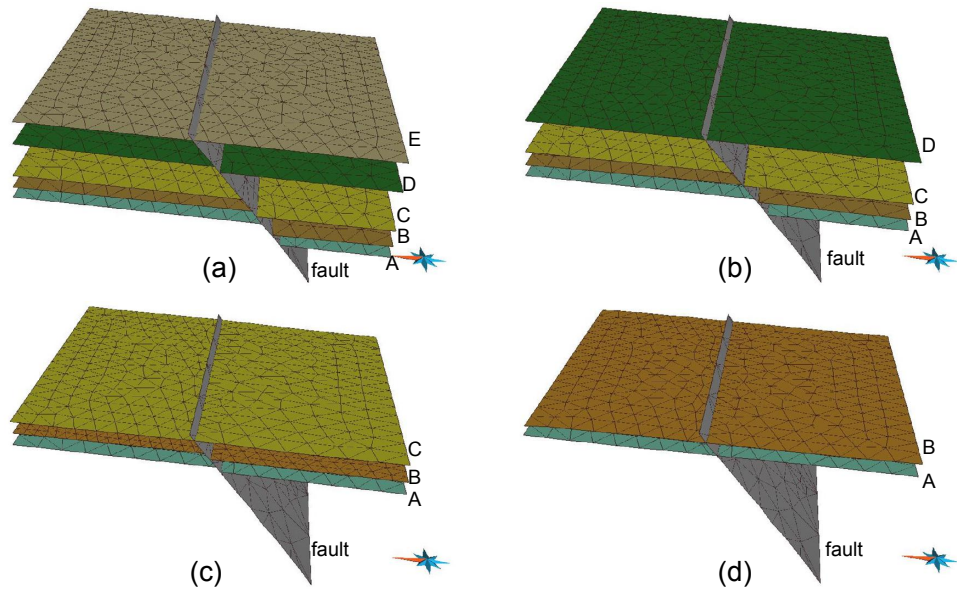
By modelling the geometry of the salt-dome as a function of time, the software can quickly answer questions, such as “What is the shape of the salt-dome at a specific time?”. The software also models the evolution of the salt-dome by continuously querying the shape of the salt-dome at consecutive time points.



**Figure 7.1:** A salt dome in time. (a)  $t = 1$ , (b)  $t = 2/3$ , (c)  $t = 1/2$ , (d)  $t = 0$

## 7.2.2 Geometrical modeling of syn-sedimentary faulting

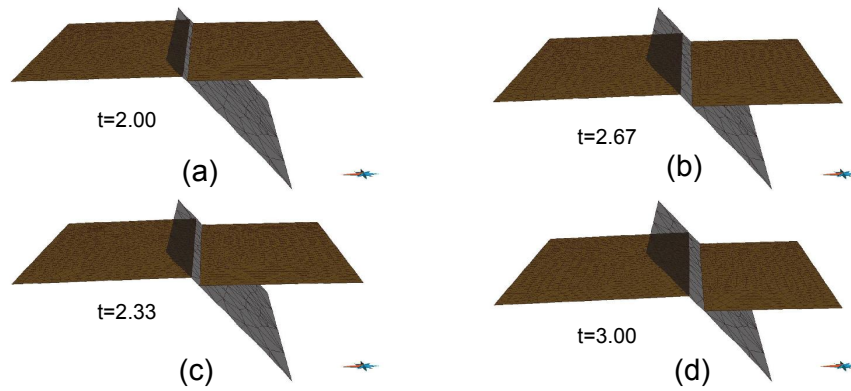
In this sample, we show the evolution of a sedimentary sequence along a normal fault which develops during sedimentation. The model was divided into a downthrown block (hanging wall) and an upthrown block (foot wall) that stays fixed. While the fault stayed in a constant location, and was active with constant



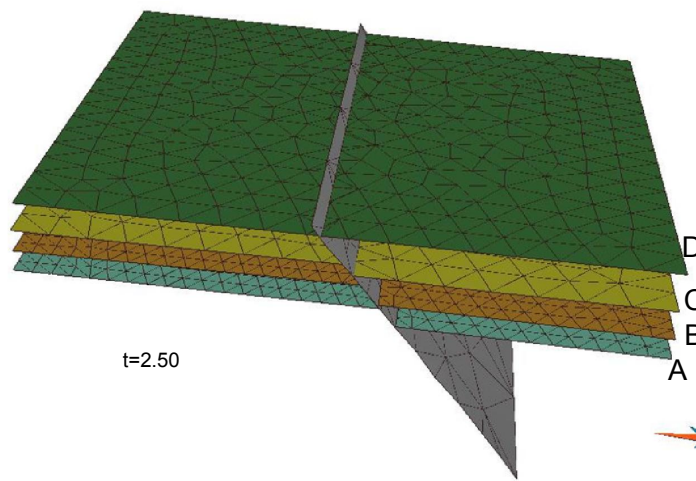
**Figure 7.2:** Given data at time instances. (a)  $t = 4$ , (b)  $t = 3$ , (c)  $t = 2$ , and (d)  $t = 1$

displacement, sedimentary units were deposited. These units were represented by their top surfaces. During their evolution, the downthrown block accommodated more sedimentary material than the time-equivalent of the upthrown block. Older sedimentary units yielded higher amounts of displacement than younger units, because they were affected by displacement for a longer time.

The given data included four sets of data at four time instances,  $t = 1$ ,  $t = 2$ ,  $t = 3$  and  $t = 4$ . At time  $t = 4$ , there were six surfaces, namely five geological horizons,  $A, B, C, D$  and  $E$ , and the fault (Figure 7.2a). At time  $t = 3$ , five surfaces represented four geological horizons,  $A, B, C$  and  $D$ , and the fault (Figure 7.2b). At time  $t = 2$ , the given data included four surfaces,  $A, B$  and  $C$ , and the fault (Figure 7.2c). At time  $t = 1$ , three surfaces represented two geological horizons  $A$  and  $B$ , and the fault (Figure 7.2d). The fault was assumed to have constant displacement throughout the time interval  $[1, 4]$ . Each horizon in each time interval was constructed by our morphological interpolation method. For example, to construct spatio-temporal data of the geological horizon,  $B$ , in time interval  $[2, 3]$  from two of its shapes at time instances  $t = 2, 3$ , Figure 7.3a and Figure 7.3d, we had to set up eight pairs of control vertices at the corners of the surfaces. Figure 7.3b and Figure 7.4c shows two intermediate surfaces of the horizon  $B$  extracted from the model at time  $t = 2.67$  and  $t = 2.33$ , respectively. By placing all of the results of the constructed data of each horizon in each time interval in the same space and time coordinates, we modeled geological processes



**Figure 7.3:** The geological horizon  $B$  in time interval  $[2, 3]$



**Figure 7.4:** The geological structure of an area of interest at time  $t = 2.50$

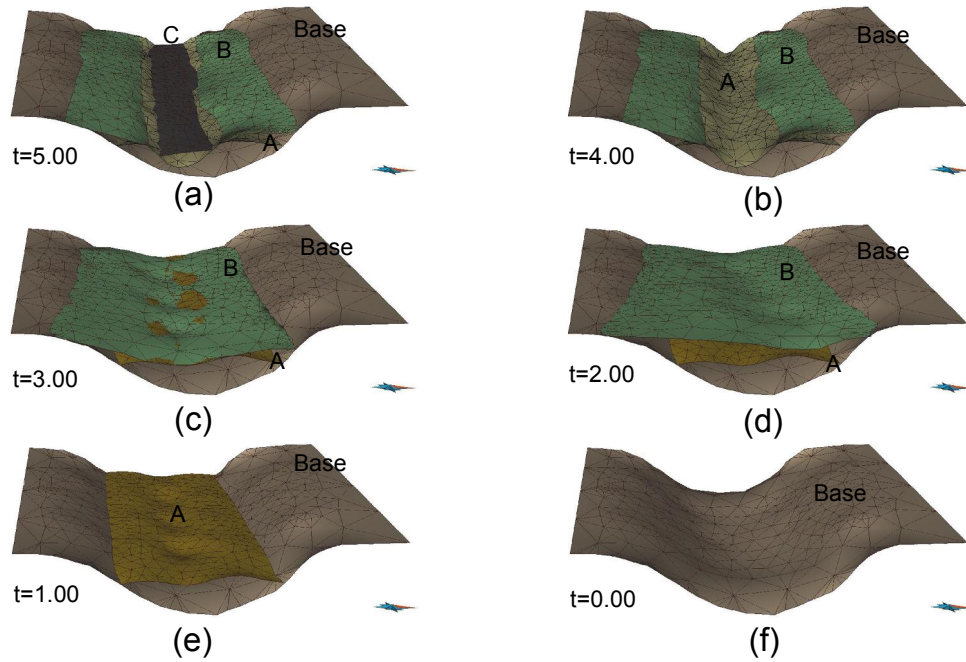
in an area and in a specific time interval. For example, the geological structure in the area of interest at time  $t = 2.50$  was calculated by the system and shown in Figure 7.4.

### 7.2.3 Modeling deposition and erosion of river sedimentary rocks

In this sample, we want to model sedimentary rocks in a river basin. River basins are characterized by erosion as well as by deposition of classic rocks with various grain size. Whether sedimentary material is eroded or deposited depends on the relation of and flow velocity. Therefore, deposited sedimentary rocks can be eroded during flooding periods.

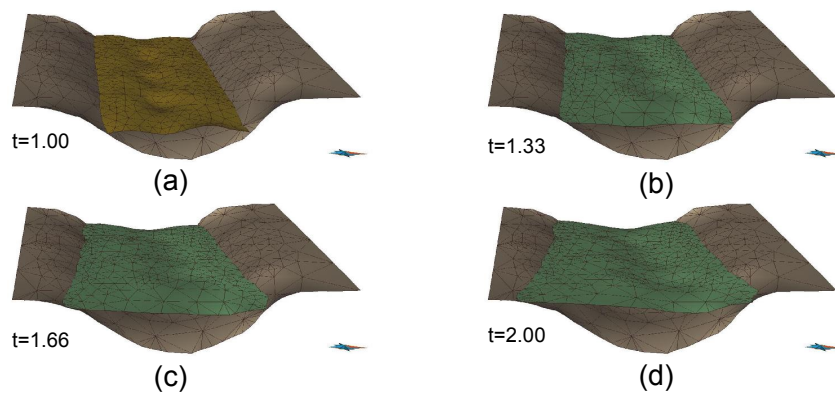
In this model, we described the temporal evolution of three sedimentary units



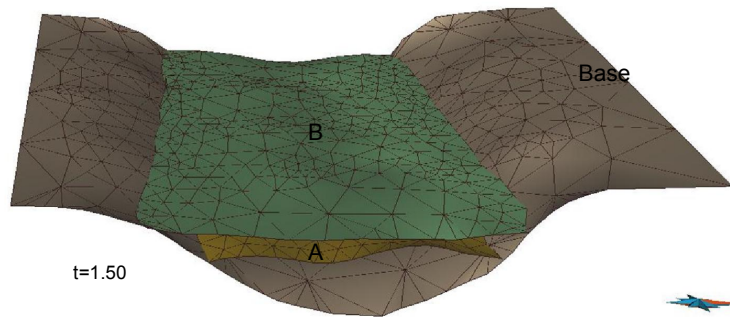


**Figure 7.5:** Given data at six time instances. (a)  $t = 5$ , (b)  $t = 4$ , (c)  $t = 3$ , (d)  $t = 2$ , (e)  $t = 1$ , and (f)  $t = 0$

each represented by its top surface. We worked with four surfaces, namely, a basement surface and top surfaces,  $A$ ,  $B$  and  $C$ , of sediments  $A$ ,  $B$  and  $C$ , respectively. The data were given at the following six time instances:  $t = 5$  (Figure 7.5a),  $t = 4$  (Figure 7.5b),  $t = 3$  (Figure 7.5c),  $t = 2$  (Figure 7.5d),  $t = 1$  (Figure 7.5e) and  $t = 0$  (Figure 7.5f). The basement surface was assumed constant. We constructed the data of the surfaces in time interval  $[0, 5]$  by constructing the data of each surface ( $A$ ,  $B$ ,  $C$ ) in each time interval,  $[0, 1]$ ,  $[1, 2]$ ,  $[3, 4]$  and  $[4, 5]$ . For brevity, we only describe the procedure to construct the data of surface  $B$  in the time interval  $[1, 2]$ . In this case, surface  $B$  at time  $t = 2$  was chose as the source mesh (Figure 7.6d), and surface  $A$  at time  $t = 1$  was chosen as the target mesh (Figure 7.6a). The controlling surface was the basement surface. Attaching constraints were used to maintain surface  $B$  in constant contact with the basement surface. Trajectories have been calculated using the finding trajectories sub-procedure, as shown in Chapter 4. By building the spatio-temporal model of the surface  $B$  in time interval  $[1, 2]$ , the surface representing the top surface of unit  $B$  can be computed at any time in between 1 and 2. For example, Figure 7.6b and Figure 7.6c show the top surfaces of unit  $B$  at time  $t = 1.33$  and  $t = 1.66$ , respectively. After constructing all of the data, the final data were placed in the same space and time coordinates to simulate certain geological processes, in this



**Figure 7.6:** Top surface B in time interval  $[1, 2]$



**Figure 7.7:** The geological structure of an area of interest at time  $t = 1.5$

case, sedimentation and erosion. Figure 7.7 depicts the geological structure in the area of interest at time  $t = 1.50$  which was queried from the system.

This sample showed that the topology between objects can change when units are deposited or eroded. For example, at time  $t = 3$  unit  $B$  was on top of unit  $A$ , but at time  $t = 4$  unit  $C$  was on top of unit  $A$ .

### 7.3 Modeling a geological structure with certain versions

In this sample, the tectonic modeling aimed to construct a deformed horizon surface in 3D and to perform a surface-restoration to the pre-deformational state. During restoration, unfolding and unfauling operations were performed, and the dilatancy of the surface was computed for each node. Depending on the modeling purpose, dilatancy can indicate zones of mineralization or zones with microfractures, such that restoration is an important interpretation step in mineral and

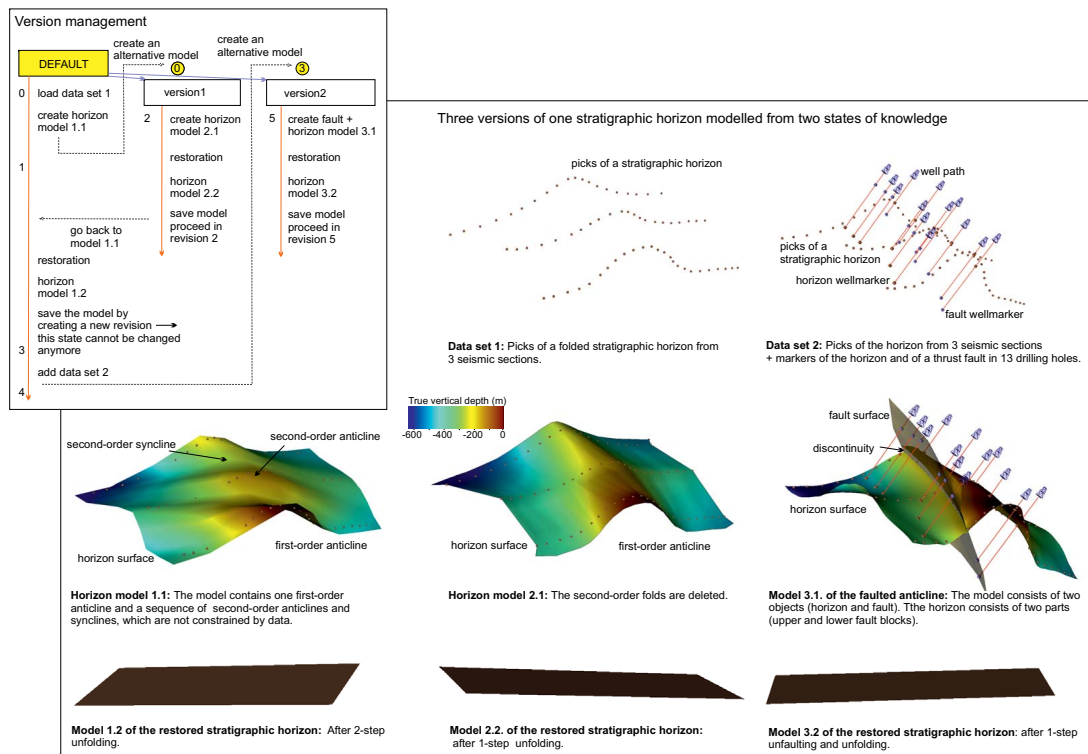


energy exploration. Two data sets were available to model: Data set 1 is comprised of 3 parallel seismic profiles, from which the stratigraphic horizon was picked and digitized. Data set 2 consists of 13 drillings, which provide well-markers for the stratigraphic horizon and a thrust fault. The 3D modeling was first performed using the seismic profiles. The drilling data set was added later, such that the models represent two states of knowledge.

We started with data set 1, which suggests that the horizon has an anticlinal structure. By default, data set 1 was added in version DEFAULT with revision 0. In this version, horizon model 1.1 was created (Figure 7.8). This horizon model shows a first-order anticline and a series of second-order anticlines and synclines with fault axes almost perpendicular to the first-order anticline. However, the second-order structure is not constrained by data and may be an artifact. The modeler can make three decisions: delete the horizon model 1.1 and proceed with the version DEFAULT revision 0, create a revision of this model, meaning he can come back to the model to see it but cannot revise it; after saving the revision, the modeler can delete the horizon surface and proceed with version DEFAULT revision 1, or the modeler can create a new version, come back to this model and work with it later. We opted for the last option because the second-order folds were not constrained by data, and we could not be sure whether the model was wrong or not. We created a version 1 to model the same geologic object (deformed stratigraphic horizon) using another geological concept (first-order fold only). Additionally, we get the opportunity to work with two models for one geologic object. When we created *version1*, revision 0 was used as the initial revision. Revisions 1 and 2 were automatically created in version DEFAULT and *version1*, respectively. Revision 2 became the working revision of *version1*. We also created a new horizon model 2.1 without second-order folds. After having modeled the horizon model 2.1, we restored it and created a restored horizon model 2.2. By doing this, we stayed in *version1* revision 2 and saved the revision after the restoration was finished. Finally, we decided to compare the results of the restoration of horizon model 1.1 and 2.1, and went back to version DEFAULT. At this time, revision 1 was the working revision to add the restored horizon model 1.2. To finalize these works, we saved the revision, creating revision 3 in version DEFAULT.

After finishing this work, we obtained data set 2. This set of drilling data provides evidence for the existence of a thrust fault and for the absence of the

second-order folds that had formed in horizon model 1.1. We re-interpreted the 3D model, including data set 2. Since the data are independent of the modeler and his interpretations, we added them to version DEFAULT, which had reached the working revision 3. We saved this revision by creating a new version where we can build the revised model while keeping the others. When creating *version2*, revision 3 is used as the initial revision, and revision 4, revision 5 were created as working revisions in version DEFAULT and *version2*, respectively. We then created a new model of the deformed horizon, which consists of a fault and a folded horizon. The topology of the horizon differs from horizon models 1.1 and 1.2 because the horizon surface was cut along the fault into an upper and a lower fault block. We stayed in the working revision and restored the horizon by unfauling and unfolding. Finally, *version2* was saved.



**Figure 7.8:** Three versions of one stratigraphic horizon modeled using two states of knowledge

One advantage of using the database combined with the version manager is that one can query the set of models (cross-version queries) because the data are stored in a database. In our case, we can ask the following: In which model does the fault appear for the first time? In which model is dilatancy highest or above a critical value? For example, the first question can be written in SQL language

as

*“select distinct (s.modelname) from surface\_ver as s where s.revision=(select min(s1.revision) from surface\_ver as s1 where s1.type='fault');”*

with further assumption that table SURFACE\_VER has the TYPE column to define the type of a surface, e.g. fault or horizon, and the MODELNAME column to define the name of the model in which a surface belong to.

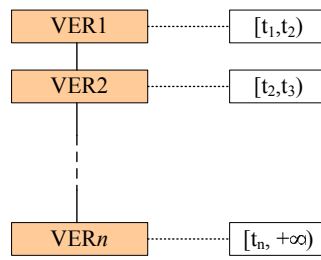
## 7.4 Managing frequently updated models

The goal of this sample was to manage geological models which are frequently updated to reflect new surveyed data during mining excavation. At the beginning of mine, a numerous boreholes were drilled to achieve data in the interesting area. An initial model representing several horizons and faults were interpreted from these borehole data. The model was primary data used to make a mining plan. During the mining excavation, the exposed surfaces, i.e., cross section at the mine positions were surveyed frequently to present the actual geological conditions. The new data presented the deviation of the model, and therefore the model must have been revised for adjusting the mining plan and obtaining a better understanding of the geological structures of the area.

Using TGSIS database versioning, such a model was considered a model with certain versions. A database version was created when new data were obtained. Then, the model was revised by re-interpreting. The revised model reflected the knowledge of modelers and was believed to be the best representation of geological conditions in the period from the time it was created to the time it was substituted by another revised model. Therefore, database versions were organized as a linear sequence and each version associated with a time period  $[starttime, endtime)$  which is closed at the *starttime* and opened at *endtime*. *Endtime* could be infinity to present that the version is believed to be true forever. Figure 7.9 shows the version sequence.

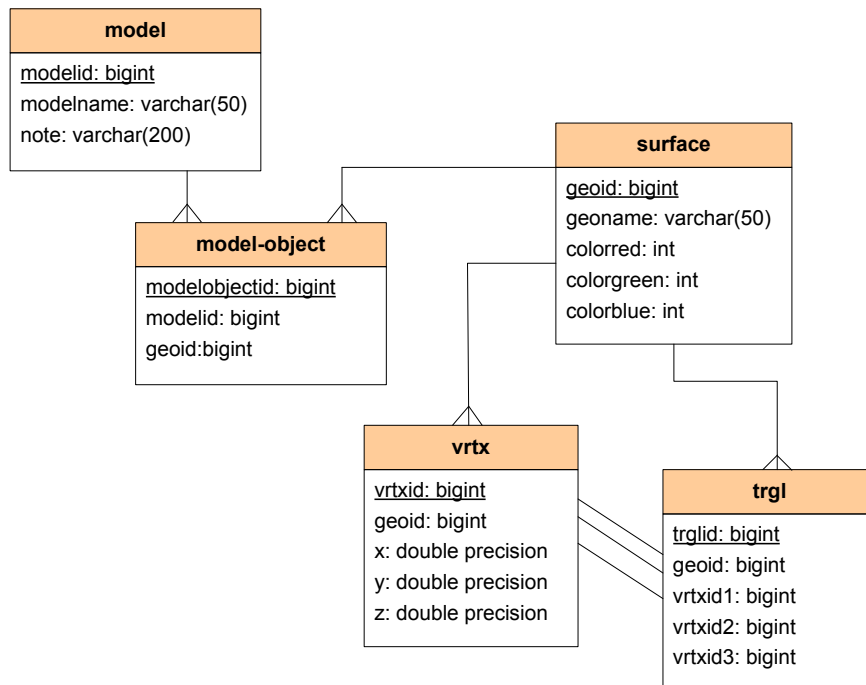
Using database versioning, all historical stages of a model, i.e. old models, are storage without data duplication. Queries across multiple versions can be constructed to compare versions of a model.

In this sample, we limited ourselves in geo-objects whose geometries are sur-



**Figure 7.9:** A linear sequence of versions

faces. Moreover, a model is a set of geo-objects and a geo-object can belong to several models. Database schema is shown in Figure 7.10.



**Figure 7.10:** Database schema of the sample

By mining excavation, certain parts of the subsurface have gone away. Depending on the purpose of the model excavated parts can be cut out from the revised model to reflect the reality or kept with the revised model for analysing. In this sample, to be easier for modeling and making mining plan, the revised models were not cut. One of the difficulties in the operation of this sample is that sometimes modelers must interpret a new surface (horizon or fault) from an only cross section. Such surfaces are strictly depended on the knowledge of the modelers.

In this sample, we did not considered aspects of data uncertainty. If consider

data uncertainty, a model can be divided into two parts: excavated parts and non-excavated parts. In excavated parts, data are certainty, i.e. full trust. Data in non-excavated parts are associated with uncertainty because of measurement errors and the uncertainty of interpretation and prediction methods. In another approach, this sample can use the approach of larger dimensions in which new versions of a model are considered along the “knowledge” dimension and the uncertainty are another dimension. The multiple indexed data model can be used in this approach.



# Chapter 8:

## Conclusions and Recommendations

---

### 8.1 Conclusions

Previous chapters have documented our work to study and contribute to the development of geoscience information systems (GSIS). The current GSIS, i.e. geomodelling or 3D computer mapping, were investigated. We found that geoscientists are interested in both geological volumes and geological variables. Geological volumes create discontinuities with complex geometries that are results of many geological processes, such as faulting, erosion, deposition. Geological variables, such as mineral grades, contaminant concentrations, hydraulic conductivities are continuous variable within the volume, but discontinuous across boundaries. The geometry of an irregular geological volume can be represented by its boundaries using so-called boundary representations (B-Reps) or implicit functions. The continuous spatial variation of a geological variable can be discretized inside the volume by regular or irregular meshes. Moreover, to fast access and process the geometry of an object, the topology relationships between object's components are always managed explicitly. Finally, GSIS manage geological objects (geo-objects) with geometry, topology and properties.

Beyond the current GSIS, which are personal desktop applications, GSIS are required to be integrated information systems by using the database technology and supporting interoperability. To achieve this, our work has reviewed the spatial database technology and standards for data exchange. Furthermore, two

database systems, namely Oracle spatial and PostgreSQL/PostGIS, which support 3-dimensional data, have also been reviewed. Finally, the work has proposed an object-relational data model whose similar data model has been implemented in the GST-Framework.

Our work has also contributed to the development of GSIS in the aspect of time integration. We have reviewed temporal GIS systems and studied ideas to represent time dimension. Time can be considered either one dimension (valid time) or multi-dimension. One time dimension can be considered as an independent dimension which strictly differ from spatial dimensions or a larger spatial dimension. When an object evolves in time, its geometry, and properties can be represented as discrete or continuous functions (of time). A number of data models have been proposed including the so-called TGSIS data model. The TGSIS data model is a general data model that represents geological objects in the domain  $R^m \times \text{time}$ . This data model contains the following features: (i) the combinatorial topological GMap model, (ii) an embedding model best adapted to linear geometries, (iii) assignment of geoscience properties to cells of various dimensions, (iv) instantaneous evolution of the topology, and (v) continuous evolution of the geometry and properties. Because GMap is not restricted in terms of dimension, GMap was also used in the data model for multiply indexed geoscience data, in which multi-dimensional time and other non-spatial dimensions are considered as larger spatial dimensions.

Capturing temporal geological data leads to the issue of modeling 4D data from certain 3D cross-sectional data. To solve this issue, morphological interpolation methods were investigated. Two primary approaches, namely mathematical morphology and the parameterisation approach, were reviewed. In this work, we proposed a method based on parameterisation techniques and adapted it for geological surfaces. This method considers only geometrical constraints, and therefore is useful for the construction of intermediate stages of a geological object when no physical or mechanical process models are available or when there is insufficient data for these models.

A characteristic of geological models is that they are always changed (updated) when having some new data. Especially in mining applications, new cross-sectional data are frequently received, so the model is changed frequently, too. These changes may be thought as the changes along the “knowledge time” axis, that means the changes of our knowledge about the reality in time. To support



this kind of data, long transactions with their benefits in TGSIS have been studied. The database versioning is a method to have the long transaction feature in the database management systems. We proposed a data schema, functions, triggers, and views to implement the database versioning, long transactions in the PostgreSQL. The database versioning in turn leads to the issue of comparison of two triangulated meshes. Our work solved this issue by proposing an algorithm which runs in time  $O(n\log(n))$  and in space  $O(n)$ .

A prototypical software has been implemented and tested with some running examples. The software proved the effectiveness of the data models, the algorithm of data construction, and the database versioning. The applicability of TGSIS in the geosciences has been also demonstrated through a number of sample applications.

## 8.2 Recommendations

Although this study had certain contributions to the development of GSIS/TGSIS, especially in the aspect of time integration. GSIS/TGSIS require a long-term development strategy and a science behind them. Many issues must be solved and improved. These include, but are not limited to tools for import/export data from/to other systems, theories and tools for user-guided geological model building including implicit modeling, operations for data querying, analysing, visualising, and output. In the near future, some algorithms and solutions should be proposed to

- capture data from numerical simulation systems,
- user-guided build geological models from raw data,
- query using topology, geometry, and properties,
- accelerate queries using index access methods, and
- perform “geoprocessing tools”, such as intersection, cut, and buffer.

In another direction, the following researches can be considered:

- extending a database management system by providing spatio-temporal data structures, methods for evaluating operations, specialised indices and join algorithms and

- studying other time dimensions, scale, and uncertainty in an unified approach.

# References

---

- Abdul-Rahman, A., Pilouk, M., 2008. Spatial Data Modelling for 3D GIS. Springer, Berlin.
- Adya, A., Liskov, B., O’Neil, P., 2000. Generalized isolation level definitions, in: Data Engineering, 2000. Proceedings. 16th International Conference on, IEEE. pp. 67–78.
- Alexa, M., 2000. Merging polyhedral shapes with scattered features. The Visual Computer 16, 26–37. URL: <http://dx.doi.org/10.1007/PL00007211>, doi:10.1007/p100007211.
- Allen, B., Curless, B., Popović, 2003. The space of human body shapes: reconstruction and parameterization from range scans, in: SIGGRAPH ’03 ACM SIGGRAPH 2003 Papers, ACM. pp. 587–594. doi:10.1145/1201775.882311.
- Apel, M., 2004. A 3d geoscience information system framework. Ph.D. thesis. Cotutelle de thèse, TU Bergakademie Freiberg, Germany, and ENSG.
- Baumgart, B.G., 1972. Winged edge polyhedron representation. Stanford University, Stanford, CA, USA. URL: <http://www.dtic.mil/dtic/tr/fulltext/u2/755141.pdf>. (accessed 02 June 2014).
- Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM 18, 509–517.
- Berenson, H., Bernstein, P., Gray, J., Melton, J., O’Neil, E., O’Neil, P., 1995. A critique of ansi sql isolation levels, in: ACM SIGMOD Record, ACM. pp. 1–10.
- Beucher, S., 1994. Interpolation d’ensembles, departitions et de fonctions. Report. Centre de Morphologie Mathematique, Ecole des Mines de Paris.
- Beucher, S., 1998. Interpolation of sets, of partitions and of functions, in: H.Heimans, J.Roedink (Eds.), Mathematical Morphology and its Applications to Image and Signal Processing. Kluwer.
- Bischoff, S., Pavic, D., Kobbelt, L., 2005. Automatic restoration of polygon models. ACM Transactions on Graphics 24, 1332–1352. doi:10.1145/1095878.1095883.

- Boissonnat, J.D., Oudot, S., 2005. Provably good sampling and meshing of surfaces. *Graphical Models* 67, 405–451. doi:10.1016/j.gmod.2005.01.004.
- Boost, 2014. Boost c++ libraries. URL: <http://www.boost.org>. (accessed 21 February 2014).
- Bors, A.G., Kechagias, L., Pitas, I., 2002. Binary morphological shape-based interpolation applied to 3-d tooth reconstruction. *IEEE Transactions on Medical Imaging* 21, 100–108. doi:10.1109/42.993129.
- Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., Levy, B., 2010. Polygon mesh processing. A K Peters, Natick, Mass.
- Brandel, S., Perrin, M., Rainaud, J.F., Schneider, S., 2001. Geological interpretation makes earth models easier to build. URL: [http://liris.cnrs.fr/~sbrandel/old\\_site/recherche/articles/Article\\_Amsterdam\\_5.3.pdf](http://liris.cnrs.fr/~sbrandel/old_site/recherche/articles/Article_Amsterdam_5.3.pdf). (accessed 02 June 2014).
- Brandel, S., Schneider, S., Perrin, M., Guiard, N., Rainaud, J.F., Lienhard, P., Bertrand, Y., 2005. Automatic building of structured geological models. *Journal of Computing and Information Science in Engineering* 5, 138. doi:10.1115/1.1884145.
- Breunig, M., Zlatanova, S., 2011. 3d geo-database research: Retrospective and future directions. *Computers & Geosciences* 37, 791–803. doi:10.1016/j.cageo.2010.04.016.
- Brisson, E., 1989. Representing geometric structures in d dimensions: topology and order, in: SCG '89 Proceedings of the fifth annual symposium on Computational geometry, ACM. pp. 218–227. doi:10.1145/73833.73858.
- Brisson, E., 1993. Representing geometric structures in d dimensions: Topology and order. *Discrete & Computational Geometry* 9, 387–426. URL: <http://dx.doi.org/10.1007/BF02189330>, doi:10.1007/bf02189330.
- Cahill, M.J., 2009. Serializable isolation for snapshot databases. Ph.D. thesis. University of Sydney.
- Cahill, M.J., Röhm, U., Fekete, A.D., 2008. Serializable isolation for snapshot databases, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA. pp. 729–738. doi:10.1145/1376616.1376690.
- Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., Evans, T.R., 2001. Reconstruction and representation of 3d objects with radial basis functions, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM. pp. 67–76.
- Caumon, G., 2010. Towards stochastic time-varying geological modeling. *Mathematical Geosciences* 42, 555–569.

- Caumon, G., Collon-Drouaillet, P., Le Carlier de Veslud, C., Viseur, S., Sausse, J., 2009. Surface-based 3d modeling of geological structures. *Mathematical Geosciences* 41, 927–945. doi:10.1007/s11004-009-9244-2.
- Caumon, G., Gray, G., Antoine, C., Titeux, M.O., 2013. Three-dimensional implicit stratigraphic model building from remote sensing data on tetrahedral meshes: Theory and application to a regional model of la popa basin, ne mexico. *Geoscience and Remote Sensing, IEEE Transactions on* 51, 1613–1621.
- Caumon, G., Lepage, F., Sword, C.H., Mallet, J.L., 2004. Building and editing a sealed geological model. *Mathematical Geology* 36, 405–424. doi:10.1023/B:MATG.0000029297.18098.8a.
- CGAL, 2014. Computational geometry algorithms library. URL: <http://www.cgal.org/>. (accessed 20 February 2014).
- Chilès, Jean-Paul., D.P., 2012. *Geostatistics : modeling spatial uncertainty*. Wiley, Hoboken, N.J.
- Clarkson, K.L., 2006. Nearest-neighbor searching and metric space dimensions. *Nearest-neighbor methods for learning and vision: theory and practice* , 15–59.
- Couclelis, H., 1992. People manipulate objects (but cultivate fields): beyond the raster–vector debate in gis, in: *Theories and methods of spatio–temporal reasoning in geographic space*. Springer, pp. 65–77.
- Cousty, J., Najman, L., Serra, J., 2009. Some morphological operators in graph spaces, in: *ISMM '09 Proceedings of the 9th International Symposium on Mathematical Morphology and Its Application to Signal and Image Processing*, Springer-Verlag. pp. 149–160. doi:10.1007/978-3-642-03613-2\_14.
- Cowan, E., Beatson, R., Ross, H., Fright, W., McLennan, T., Evans, T., Carr, J., Lane, R., Bright, D., Gillman, A., 2003. Practical implicit geological modelling, in: *Fifth International Mining Geology Conference*, pp. 17–19.
- Datar, M., Immorlica, N., Indyk, P., Mirrokni, V., 2004. Locality-sensitive hashing scheme based on p-stable distributions, in: *SCG '04 Proceedings of the twentieth annual symposium on Computational geometry*, ACM New York, NY, USA ©2004. pp. 253–262. doi:10.1145/997817.997857.
- De Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.C., 2000. *Computational geometry: algorithms and applications*. 2 ed., Springer.
- Dhanabal, S., Chandramathi, S., 2011. Review of various k-nearest neighbor query processing techniques. *International Journal of Computer Applications* 31, 14–22.
- Dias, F., Cousty, J., Najman, L., 2011. Some morphological operators on simplicial complex spaces, in: *DGCI'11 Proceedings of the 16th IAPR international conference on Discrete geometry for computer imagery*, Springer-Verlag. pp. 441–452.

- Dias, F., Cousty, J., Najman, L., 2014. Dimensional operators for mathematical morphology on simplicial complexes. CoRR abs/1401.5602.
- Erwig, M., Güting, R.H., Schneider, M., Vazirgiannis, M., 1999. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. *GeoInformatica* 3, 269–296. doi:10.1023/a:1009805532638.
- ESRI, 2014a. The arcgis 3d analyst. URL: <http://www.esri.com/software/arcgis/extensions/3danalyst>. (accessed 15 February 2014).
- ESRI, 2014b. The arcgis software family. URL: <http://www.esri.com/>. (accessed 15 February 2014).
- ESRI, 2014c. Arcsde 10 developer help. URL: <http://help.arcgis.com/en/geodatabase/10.0/sdk/arcsde/welcome.htm>. (accessed 15 February 2014).
- ESRI, 2014d. Esri cityengine. URL: <http://www.esri.com/software/cityengine>. (accessed 20 June 2014).
- Floater, M., Hormann, K., Kós, G., 2006. A general construction of barycentric coordinates over convex polygons. *Advances in Computational Mathematics* 24, 311–331. URL: <http://dx.doi.org/10.1007/s10444-004-7611-6>, doi:10.1007/s10444-004-7611-6.
- Floater, M.S., 1997. Parametrization and smooth approximation of surface triangulations. *Comput. Aided Geom. Des.* 14, 231–250. doi:10.1016/S0167-8396(96)00031-3.
- Floater, M.S., 1998. Parametric tilings and scattered data approximation. *Int. J. Shape Model.* 04, 165–182. doi:10.1142/S021865439800012X.
- Floater, M.S., 2003. Mean value coordinates. *Comput. Aided Geom. Des.* 20, 19–27. doi:10.1016/S0167-8396(03)00002-5.
- Floater, M.S., Gotsman, C., 1999. How to morph tilings injectively. *J. Comput. Appl. Math.* 101, 117–129. doi:10.1016/S0377-0427(98)00202-7.
- Floriani, L.D., Mesmoudi, M.M., Morando, F., Puppo, E., 2002. Non-manifold decomposition in arbitrary dimensions, in: *DGCI '02 Proceedings of the 10th International Conference on Discrete Geometry for Computer Imagery*, Springer-Verlag. pp. 69–80.
- Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M., 2000. A data model and data structures for moving objects databases, in: Dunham, M., Naughton, J.F., Chen, W., Koudas, N. (Eds.), the 2000 ACM SIGMOD international conference. ACM, pp. 319–330.
- Frank, T., Tertois, A.L., Mallet, J.L., 2007. 3d-reconstruction of complex geological interfaces from irregularly distributed and noisy point data. *Computers & Geosciences* 33, 932–943.

- Gabriel, P., Gietzel, J., Le, H.H., Schaeben, H., 2012. Komponenten einer 3d gdi. *gis.Science Ausgabe 4/2012*. Wichmann Verlag, Journal 4/2012, 155–160 URL: <http://www.wichmann-verlag.de/gis-fachzeitschriften/artikelarchiv/2012/gis-science-ausgabe-04-2012/komponenten-einer-3d-gdi.html>.
- Garcia-Molina, H., Ullman, J.D., Widom, J., 2009. Database systems: The complete book. 2nd ed., Pearson Prentice Hall, Upper Saddle River, N.J.
- GEOS, 2014. Geos - geometry engine, open source. URL: <http://trac.osgeo.org/geos/>. (accessed 17 February 2014).
- GiGa-Infosystems, 2014. Gst® geosciences in space and time. URL: <http://www.giga-infosystems.com/>. (accessed 17 February 2014).
- gOcad, 2014. Gocad research group. URL: <http://www.gocad.org>. (accessed 18 February 2014).
- Goodchild, M.F., 2010. Twenty years of progress: Giscience in 2010. *Journal of Spatial Information Science* 1, 3–20.
- Goodchild, M.F., Yuan, M., Cova, T.J., 2007. Towards a general theory of geographic representation in gis. *International journal of geographical information science* 21, 239–260.
- Greiner, G., Hormann, K., 1997. Interpolating and Approximating Scattered 3D-data with Hierarchical Tensor Product B-Splines. *Vanderbilt University Press*. pp. 163–172.
- Guennebaud, G., Jacob, B., others., 2010. Eigen v3. URL: <http://eigen.tuxfamily.org>. (accessed 18 February 2014).
- Güting, R.H., Schneider, M., 2005. Moving objects databases. *Morgan Kaufmann series in data management systems*, Morgan Kaufmann, San Francisco, Calif.; London.
- HDF5, 2014. Hierarchical data format (hdf) version 5. URL: <http://www.hdfgroup.org/HDF5/>. (accessed 17 February 2014).
- Heimans, H.J.A.M., Ronse, C., 1990. The algebraic basis of mathematical morphology. i. dilations and erosions. *Comput. Vision Graph. Image Process.* 50, 245–295. doi:10.1016/0734-189x(90)90148-o.
- Hormann, K., Greiner, G., 2000. MIPS: An efficient global parametrization method. Report. DTIC Document.
- Hormann, K., Polthier, K., Sheffer, A., 2008. Mesh parameterization: Theory and practice, in: *SIGGRAPH ASIA 2008 Course Notes*, pp. 1–87.
- Houlding, S.W., 1994. 3D Geoscience Modeling Computer Techniques for Geological Characterization. Springer.

- Iwanowski, M., 2014. Morphological interpolation. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.4506&rep=rep1&type=pdf>. (accessed 14 February 2014).
- van Kaick, O., Zhang, H., Hamarneh, G., Cohen-Or, D., 2011. A survey on shape correspondence. *Comput. Graph. Forum* 30, 1681–1707. doi:10.1111/j.1467-8659.2011.01884.x.
- Kaufman, A., 1987. Efficient algorithms for 3d scan-conversion of parametric curves, surfaces, and volumes. *SIGGRAPH Comput. Graph.* 21, 171–179. doi:10.1145/37402.37423.
- Kelk, B., 1991. 3d modelling with geoscientific information systems: The problem, in: Turner, A. (Ed.), *Three-Dimensional Modeling with Geoscientific Information Systems*. Kluwer Academic Publishers, Dordrecht, The Netherlands. volume 354, pp. 29–38.
- Kettner, L., 1999. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry* 13, 65–90. doi:10.1016/S0925-7721(99)00007-3.
- Kjenstad, K., 2006. On the integration of object-based models and field-based models in gis. *International Journal of Geographical Information Science* 20, 491–509.
- Kobbelt, L.P., Botsch, M., Schwaner, U., Seidel, H.P., 2001. Feature sensitive surface extraction from volume data, in: Pocock, L. (Ed.), *SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive*. ACM New York, NY, USA, pp. 57–66.
- Kraevoy, V., Sheffer, A., 2004. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.* 23, 861–869. doi:10.1145/1015706.1015811.
- Le, H.H., 2013. Spatio-temporal data construction. *ISPRS International Journal of Geo-Information* 2, 837–853. doi:10.3390/ijgi2030837.
- Le, H.H., Gabriel, P., Gietzel, J., Schaeben, H., 2013. An object-relational spatio-temporal geoscience data model. *Computers & Geosciences* 57, 104–115. doi:10.1016/j.cageo.2013.04.014.
- Le, H.H., Schaeben, H., Jasper, H., Görz, I., 2014 (accepted). Database versioning and its implementation in geoscience information systems. *Computers & Geosciences* doi:10.1016/j.cageo.2014.05.011.
- Lévy, B., Mallet, J., 1999. Cellular modeling in arbitrary dimension using generalized maps. URL: [http://alice.loria.fr/publications/papers/1999/g\\_maps/g\\_maps.pdf](http://alice.loria.fr/publications/papers/1999/g_maps/g_maps.pdf). (accessed 16 May 2012).
- Liaw, Y.C., Leou, M.L., Wu, C.M., 2010. Fast exact k nearest neighbors search using an orthogonal search tree. *Pattern Recognition* 43, 2351–2358.



- Lienhardt, P., 1989. Subdivisions of  $n$ -dimensional spaces and  $n$ -dimensional generalized maps, in: SCG '89 Proceedings of the fifth annual symposium on Computational geometry, ACM. pp. 228–236. doi:10.1145/73833.73859.
- Lienhardt, P., 1994.  $N$ -dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry & Applications* 04, 275–324. URL: <http://www.worldscientific.com/doi/abs/10.1142/S0218195994000173>, doi:doi:10.1142/S0218195994000173.
- Lienhardt, P., Fuchs, L., Bertrand, Y., 2009. Combinatorial models for topology-based geometric modeling. URL: <http://hal-unilim.archives-ouvertes.fr/docs/00/58/07/08/PDF/8qm22.pdf>. (accessed 16 May 2012).
- Liu, Y., Yan, H., Martin, R., 2011. As-rigid-as-possible surface morphing. *J. Comput. Sci. Technol.* 26, 548–557. doi:10.1007/s11390-011-1154-3.
- Lorensen, W.E., Cline, H.E., 1987. Marching cubes: A high resolution 3d surface construction algorithm, in: Stone, M.C. (Ed.), *ACM SIGGRAPH Computer Graphics*, ACM New York, NY, USA. pp. 163–169.
- Mallet, J.L., 2002. *Geomodeling*. Applied geostatistics series, Oxford University Press, Oxford; New York.
- Meyer, F., 1994a. Interpolations. Report. Centre de Morphologie Mathematique, Ecole des Mines de Paris.
- Meyer, F., 1994b. Morphological interpolation of mosaic images, in: P.Maragos (Ed.), *Mathematical Morphology and its Applications to Image and Signal Processing*. Kluwer.
- Natali, M., Lidal, E.M., Parulek, J., Viola, I., Patel, D., 2013. Modeling terrains and subsurface geology, in: *Eurographics 2013-State of the Art Reports*, The Eurographics Association. pp. 155–173.
- OGC, 2010. *Opengis implementation standard for geographic information – simple feature access – part 2: Sql option (06-104r4)*. URL: <http://www.opengeospatial.org/standards/sfs>. (accessed 16 May 2012).
- OGC, 2011. *Opengis implementation standard for geographic information – simple feature access – part 1: Common architecture (06-103r4)*. URL: <http://www.opengeospatial.org/standards/sfa>. (accessed 16 May 2012).
- Ohori, K.A., Biljecki, F., Stoter, J., Ledoux, H., 2013a. Manipulating higher dimensional spatial information, in: *Proceedings of the 15th AGILE International Conference on Geographic Information Science - Geographic Information Science at the Heart of Europe*, AGILE. pp. 1–7.
- Ohori, K.A., Ledoux, H., 2013. Using extrusion to generate higher-dimensional gis datasets, in: *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM. pp. 398–401.

- Ohori, K.A., Ledoux, H., Stoter, J., 2013b. Modelling higher dimensional data for gis using generalised maps, in: Computational Science and Its Applications–ICCSA 2013. Springer, pp. 526–539.
- van Oosterom, P., Stoter, J., 2010. 5d data modelling: full integration of 2d/3d space, time and scale dimensions, in: Geographic information science. Springer, pp. 310–324.
- OpenNL, 2014. Open numerical library. URL: <http://alice.loria.fr/index.php/software/4-library/23-opennl.html>. (accessed 21 February 2014).
- Oracle, 2013a. Oracle database 12c: Workspace manager (white paper ). URL: [http://download.oracle.com/otndocs/products/workspace\\_manager/pdf/workspace\\_manager\\_12c\\_twp.pdf](http://download.oracle.com/otndocs/products/workspace_manager/pdf/workspace_manager_12c_twp.pdf). (accessed 15 February 2014).
- Oracle, 2013b. Oracle Database: Workspace Manager Developer’s Guide 12c Release 1 (12.1) - E17893-07. Oracle.
- Oracle, 2013c. Oracle ®Spatial Developer’s Guide 11g Release 2 (11.2): E11830-14. Oracle. URL: [http://docs.oracle.com/cd/E11882\\_01/appdev.112/e11830/toc.htm](http://docs.oracle.com/cd/E11882_01/appdev.112/e11830/toc.htm).
- Paradigm-GOCAD, 2014. Gocad – the framework for subsurface modeling. URL: <http://www.pdgm.com/products/GOCAD>. (accessed 16 February 2014).
- Peuquet, D.J., 1994. It’s about time: A conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of american Geographers* 84, 441–461.
- Pinkall, U., Juni, S.D., Polthier, K., 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics* 2, 15–36.
- Polthier, K., Rumpf, M., 1995. A concept for time-dependent processes, in: Göbel, M., Müller, H., Urban, B. (Eds.), *Visualization in Scientific Computing*. SpringerVerlag, pp. 137–153.
- Ports, D.R., Grittner, K., 2012. Serializable snapshot isolation in postgresql. *Proceedings of the VLDB Endowment* 5, 1850–1861.
- PostGIS, 2014. Spatial and geographic objects for postgresql. URL: <http://postgis.net/>. (accessed 17 February 2014).
- PostgreSQL, 2014. Postgresql - the world most advanced open source database. URL: <http://www.postgresql.org/>. (accessed 16 February 2014).
- Praun, E., Sweldens, W., Schröder, P., 2001. Consistent mesh parameterizations, in: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’01*, Los Angeles, CA, USA, 12-17 August, ACM Press: New York, NY, USA. pp. 179–184.

- Qi, L., Schneider, M., 2012. Monet: modeling and querying moving objects in spatial networks, in: Ali, M., Banaei-Kashani, F., Hoel, E. (Eds.), the Third ACM SIGSPATIAL International Workshop. ACM, pp. 48–57.
- Ravada, S., Kazar, B., Kothuri, R., 2009. Query processing in 3d spatial databases: Experiences with oracle spatial 11g, in: Lee, J., Zlatanova, S. (Eds.), 3D Geo-Information Sciences. Springer Berlin Heidelberg. Lecture Notes in Geoinformation and Cartography, pp. 153–173. URL: [http://dx.doi.org/10.1007/978-3-540-87395-2\\_10](http://dx.doi.org/10.1007/978-3-540-87395-2_10), doi:10.1007/978-3-540-87395-2\_10.
- Raza, A., 2012. Working with spatio-temporal data type. Esri, 380 New York Street, Redlands, California 92373-8100 USA. pp. XXXIX–B2:5–10.
- RESQML, 2012. Resqml version 1.1 specifications. URL: <http://www.energistics.org/reservoir/resqml-standards/current-standards>. (accessed 14 November 2012).
- Samet, H., 1994. The Design and Analysis of Spatial Data Structures. Addison–Wesley Pub.
- Schneider, M., 2009. Moving objects in databases and gis: State-of-the-art and open problems, in: Navratil, G. (Ed.), Research Trends in Geographic Information Science. Springer Berlin Heidelberg. Lecture Notes in Geoinformation and Cartography, pp. 169–187. URL: [http://dx.doi.org/10.1007/978-3-540-88244-2\\_12](http://dx.doi.org/10.1007/978-3-540-88244-2_12), doi:10.1007/978-3-540-88244-2\_12.
- Schreiner, J., Asirvatham, A., Praun, E., Hoppe, H., 2004. Inter-surface mapping. ACM Trans. Graph. 23, 870–877. doi:10.1145/1015706.1015812.
- Serra, J., 1983. Image Analysis and Mathematical Morphology. Academic Press, Inc.
- Serra, J., 1994. Interpolations et distance de Hausdorff. Report. Centre de Morphologie Mathématique, Ecole des Mines de Paris.
- Serra, J., 1998. Hausdorff distance and interpolations, in: H.Heimans, J.Roedink (Eds.), Mathematical Morphology and its Applications to Image and Signal Processing. Kluwer.
- Sethian, J.A., 1996. A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences 93, 1591–1595. URL: <http://www.pnas.org/content/93/4/1591.abstract>.
- Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S., 1997. Modeling and Querying Moving Objects. IEEE Computer Society, Washington, DC, USA. ICDE 97, pp. 422–432. URL: <http://dl.acm.org/citation.cfm?id=645482.653301>.
- Sistla, P., Wolfson, O., Chamberlain, S., Dao, S., 1998. Querying the Uncertain Position of Moving Objects. Springer. pp. 310–337.

- Sumner, R.W., Popović, J., 2004. Deformation transfer for triangle meshes, in: ACM Transactions on Graphics (TOG) – Proceedings of ACM SIGGRAPH 2004, ACM. pp. 399–405. doi:10.1145/1186562.1015736.
- Toledo, S., Chen, D., Rotkin, V., 2014. Taucs: A library of sparse linear solvers. URL: <http://www.tau.ac.il/~stoledo/taucs/>. (accessed 14 February 2014).
- Trusheim, F., 1960. Mechanism of salt migration in northern germany. AAPG Bulletin 44, 1519–1540.
- Turner, A., 2000. Geoscientific modeling: past, present, and future. Geographic information systems in petroleum exploration and development. AAPG computer applications in geology 4, 27–36.
- Turner, A.K., 1991. Three-dimensional modeling with geoscientific information systems. volume 354. Springer.
- Turner, A.K., 2006. Challenges and trends for geological modelling and visualisation. Bulletin of Engineering Geology and the Environment 65, 109–127.
- Turner, A.K., Gable, C.W., 2007. A review of geological modeling. Three-dimensional geologic mapping for groundwater applications. Minnesota Geological Survey Open-file Report , 07–4.
- Tutte, W.T., 1960. Convex representations of graphs, in: Proc. London Math. Soc. (3) 10, pp. 304–320.
- Tutte, W.T., 1963. How to draw a graph, in: London Math. Soc., 13(52), pp. 743–767.
- Vendeville, B.C., Jackson, M.P., 1992. The rise of diapirs during thin-skinned extension. Marine and Petroleum Geology 9, 331–354.
- Voudouris, V., 2008. Geospatial Modelling of Indeterminate Phenomena: The Object–Field Model with Uncertainty and Semantics. Ph.D. thesis. City University London. URL: <http://dx.doi.org/10.2139/ssrn.1292262>.
- Voudouris, V., 2011. Towards a conceptual synthesis of dynamic and geospatial models: fusing the agent–based and object–field models. Environment and Planning–Part B 38, 95–114.
- Weiler, K., 1985. Edge-based data structures for solid modeling in curved-surface environments. IEEE Computer Graphics and Applications 5, 21–40. doi:10.1109/mcg.1985.276271.
- Worboys, M.F., 1994. A unified model for spatial and temporal information. The Computer Journal 37, 26–34.

- Wright, D., 2010. Giscience. URL: [http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/19044/GIScience\\_all.pdf?sequence=1](http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/19044/GIScience_all.pdf?sequence=1). (accessed 02 June 2014).
- Xu, J., Güting, R.H., 2013. A generic data model for moving objects. *GeoInformatica* 17, 125–172. doi:10.1007/s10707-012-0158-7.
- Yan, H., Hu, S., Martin, R., 2007. 3d morphing using strain field interpolation. *J. Comput. Sci. Technol.* 22, 147–155. doi:10.1007/s11390-007-9020-z.
- Yuan, M., 2001. Representing complex geographic phenomena in gis. *Cartography and Geographic Information Science* 28, 83–96.
- Zatloukal, K., Johnson, M., Ladner, R., 2002. Nearest neighbor search for data compression, in: Goldwasser, M., Johnson, D., McGeoch, C. (Eds.), *Data Structures Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*. AMS. volume 59, pp. 69–86.
- Zlatanova, S., 2000. 3D GIS for Urban Development. Ph.D. thesis. ITC Dissertation Series No. 69, The International Institute for Aerospace Survey and Earth Sciences, The Netherlands.